



南京大学

研究生毕业论文 (申请工程硕士学位)

论 文 题 目 基于主动学习的警告排序系统的
设计与实现

作 者 姓 名 钱美缘

学 科、专 业 名 称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授 房春荣 助理研究员

2022 年 5 月 20 日

学 号：MF20320109

论文答辩日期：2022 年 5 月 20 日

指 导 教 师： (签字)

Design and Implementation of an Alarm Ranking System Based on Active Learning

by

Meiyuan Qian

Supervised by

Professor Zhenyu Chen

Assistant Professor Chunrong Fang

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of

MASTER OF ENGINEERING

in

Software Engineering



Software Institute
Nanjing University

May 20, 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于主动学习的警告排序系统的设计与实现

工程硕士（软件工程领域） 专业 2020 级硕士生姓名：钱美缘
指导教师（姓名、职称）：陈振宇 教授 房春荣 助理研究员

摘 要

静态代码分析工具通过分析词法、语法、数据流和控制流，能在不执行代码的情况下发现软件中存在的缺陷。目前，静态代码分析工具已被广泛地应用于开源软件和商业软件开发流程的软件质量保障中。然而静态代码分析工具过近似估计的程序行为，导致其产生了大量的误报警告，开发人员不得不花费大量时间逐一审核。

为了优先审核正报警告，现有的研究提出了利用警告历史信息和机器学习模型的排序方法。但是这些方法不能根据审核人员的实时反馈更新对警告的排序。此外训练机器学习模型需要大量数据集，而标注警告的正误报是件费时费力的工作。针对以往研究中存在的问题，本文提出了基于主动学习的警告排序技术，能将审核人员的标记结果用于后续的排序中。同时模型的训练集得到了更新，帮助克服了传统机器学习模型训练时数据集不足的问题。系统借助查询策略在未标记的警告中选择最有可能是正报的警告分配给审核人员审核，避免他们检查全部的警告，达到提升审核效率的目的。本文使用了 Vuejs 前端框架，Flask 后端框架，SQLite 轻量级数据库，RabbitMQ 消息队列，借助 sklearn 等机器学习包开发了基于主动学习的警告排序系统。

为了证明所提出方法的有效性，本文构建了来自于 5 个 Apache 开源项目的代码警告数据集，并在此基础上设计了对比实验。通过和现有的 5 种排序方法相比，实验结果表明本文的方法在 AUC 指标上提升了约 40%。而且经过比较，使用加入数据平衡策略的集成学习模型时实验效果更好，其 AUC 指标普遍在 0.9 以上。因此通过使用本文实现的系统，审核人员可以更快地找到正报，减轻了审核大量无意义警告的负担，及时修复代码中的缺陷。

关键词：主动学习；警告排序；机器学习；静态分析

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Design and Implementation of an Alarm Ranking System Based on Active Learning

SPECIALIZATION: Software Engineering

POSTGRADUATE: Meiyuan Qian

MENTOR: Professor Zhenyu Chen Assistant Professor Chunrong Fang

Abstract

Static code analysis tools detect software defects by analyzing the lexical, syntax, data flow, and control flow without executing source codes. Currently, static code analysis tools have been widely used in open-source software and commercial software quality assurance. However, static code analysis tools estimate the program behavior over-approximately, resulting in many false alarms, and developers have to spend lots of time reviewing them one by one.

Existing research proposes ranking methods utilizing alert history information and machine learning models to review positive alerts preferentially. But these methods cannot update the ranking of alarms via real-time feedback from reviewers. In addition, training machine learning models require large datasets, and labeling positive and false alarms are time-consuming and labor-intensive. Aiming at the problems existing in the previous research, this thesis proposes an alarm ranking technology based on active learning, which uses the marking results in the subsequent ranking. At the same time, the training set of the model has been updated, helping to overcome the problem that insufficient data set when training traditional machine learning models. With the help of the query strategy, the system selects the alarms that are most likely to be true among the unmarked alarms and assigns them to reviewers for review, preventing them from checking all alarms and improving review efficiency. This thesis uses Vuejs as the front-end framework, Flask as the back-end framework, SQLite as the lightweight database, RabbitMQ as the message queue server, and develops an alarm ranking system based on active learning with the help of machine learning packages such as scikit-learn.

To demonstrate the effectiveness of our method, we construct code alarm datasets from 5 Apache open source projects and design comparative experiments on them.

Compared with the existing five ranking methods, the experimental results show that the method in this thesis improves the AUC by about 40%. After comparison, the experimental effect is better when using the ensemble learning model with data balance strategies, and its AUC index is generally above 90%. Therefore, by using the system implemented in this thesis, reviewers can find true alarms faster, reduce the burden of reviewing many meaningless alarms, and fix defects in the code on time.

keywords: Active learning, alarm ranking, machine learning, static analysis

目 录

目 录	v
插图清单	ix
附表清单	xi
第一章 引言	1
1.1 项目背景与意义	1
1.2 国内外研究现状	3
1.2.1 静态代码分析工具	3
1.2.2 警告误报检测技术	4
1.2.3 主动学习技术	7
1.3 本文主要工作	8
1.4 本文组织结构	8
第二章 相关技术综述	11
2.1 主动学习	11
2.1.1 场景分类	11
2.1.2 查询策略	13
2.2 集成学习	13
2.2.1 Bagging 方法	14
2.2.2 Boosting 方法	15
2.2.3 Stacking 方法	15
2.3 警告特征提取	16
2.4 Flask 框架	17
2.5 Vue.js 框架	18
2.6 RabbitMQ 消息队列	19
2.7 本章小结	19
第三章 警告排序系统的需求分析与设计	21
3.1 系统整体概述	21
3.2 系统需求分析	22

3.2.1 涉众分析	22
3.2.2 功能性需求	23
3.2.3 非功能性需求	24
3.2.4 用例模型描述	25
3.3 系统架构设计	30
3.4 系统视图模型	32
3.4.1 逻辑视图	33
3.4.2 开发视图	34
3.4.3 进程视图	35
3.4.4 物理视图	35
3.5 数据库设计	37
3.6 系统详细设计	39
3.6.1 项目管理模块	40
3.6.2 特征提取模块	42
3.6.3 主动学习模块	45
3.6.4 警告审核模块	48
3.7 本章小结	50
第四章 警告排序系统的实现	51
4.1 整体概述	51
4.1.1 全局配置	51
4.1.2 文件存储	52
4.1.3 身份认证	52
4.2 项目管理模块的实现	53
4.2.1 项目上传	54
4.2.2 项目扫描与处理	54
4.3 特征提取模块的实现	57
4.3.1 代码度量特征	57
4.3.2 代码分析特征	59
4.3.3 警告自身特征	60
4.3.4 警告分布特征	62
4.3.5 特征合并	63
4.4 主动学习模块的实现	63

目 录	vii
4.4.1 相同警告预排序	64
4.4.2 机器学习模型	67
4.4.3 查询策略	68
4.4.4 整体流程	69
4.5 警告审核模块的实现	71
4.6 本章小结	71
第五章 实验设计与系统测试	73
5.1 实验设计	73
5.1.1 研究问题	73
5.1.2 数据集收集	74
5.1.3 评估指标	76
5.1.4 实验结果	77
5.2 系统测试	80
5.2.1 测试环境	80
5.2.2 功能测试	81
5.2.3 性能测试	84
5.2.4 用户界面测试	85
5.3 本章小结	89
第六章 总结与展望	91
6.1 总结	91
6.2 展望	92
致 谢	93
参考文献	95
简历与科研成果	101

插图清单

2.1	Bagging 方法示意图	14
2.2	Boosting 方法示意图	15
2.3	Stacking 方法示意图	16
3.1	系统总体流程图	21
3.2	系统用例图	26
3.3	系统架构图	31
3.4	系统逻辑视图	33
3.5	系统开发视图	34
3.6	系统进程视图	35
3.7	系统物理视图	36
3.8	系统实体类设计图	37
3.9	项目管理模块流程图	40
3.10	项目管理模块核心类图	41
3.11	项目管理模块时序图	42
3.12	特征提取模块流程图	43
3.13	特征提取模块核心类图	43
3.14	特征提取模块时序图	44
3.15	主动学习模块流程图	45
3.16	主动学习模块核心类图	46
3.17	主动学习模块时序图	47
3.18	警告审核模块流程图	48
3.19	警告审核模块核心类图	49
3.20	警告审核模块时序图	50
4.1	身份认证的实现代码	52
4.2	访问控制的实现代码	53
4.3	项目上传的实现代码	54

4.4	调用 SpotBugs 的代码	55
4.5	SpotBugs 报告示例	55
4.6	格式化文件并调整行号的实现代码	56
4.7	调用 Jhawk 提取代码度量特征的实现代码	58
4.8	代码分析特征提取的实现代码	60
4.9	警告自身特征提取的实现代码	61
4.10	特征合并的实现代码	63
4.11	基于位置匹配的实现代码	65
4.12	基于哈希值匹配的实现代码	66
4.13	BaggingClassifierModel 的实现代码	67
4.14	StackingClassifierModel 的实现代码	68
4.15	MultiplyClassifierModel 的实现代码	68
4.16	MultiplyClassifierModel 的实现代码	69
4.17	主动学习模型的实现代码	70
4.18	消息队列接收审核任务的实现代码	71
5.1	查看项目列表页面	86
5.2	项目上传页面	86
5.3	查看最终报告页面	87
5.4	查看审核任务列表页面	87
5.5	查看审核任务详情页面	88
5.6	查看审核任务历史页面	88

附表清单

3.1	涉众分析表	23
3.2	系统功能性需求列表	24
3.3	查看审核任务列表用例描述	26
3.4	查看警告信息用例描述	27
3.5	填写审核结果用例描述	27
3.6	账户注册用例描述	28
3.7	查看项目列表用例描述	28
3.8	上传项目用例描述	29
3.9	查看最终报告用例描述	29
3.10	审核人员管理用例描述	30
3.11	Worker 类设计描述	38
3.12	Alarm 类设计描述	38
3.13	Project 类设计描述	39
3.14	Label 类设计描述	39
4.1	系统全局配置	51
4.2	系统文件存储路径	52
4.3	代码度量特征表	58
4.4	代码分析特征表	59
4.5	警告自身特征表	61
4.6	警告分布特征表	62
5.1	项目详细信息	75
5.2	Maven 仓库文件命名格式	75
5.3	项目警告详细信息	76
5.4	RQ1 的实验结果	78
5.5	RQ2 的实验结果	79
5.6	测试环境物理配置	80

5.7	测试环境服务器软件配置	81
5.8	查看项目列表测试用例描述	81
5.9	项目上传测试用例描述	82
5.10	查看最终报告测试用例描述	82
5.11	查看审核任务测试用例描述	83
5.12	警告审核测试用例描述	83
5.13	查看审核任务历史测试用例描述	84
5.14	性能测试接口列表	85
5.15	性能测试接口列表	85

第一章 引言

1.1 项目背景与意义

现代社会中，计算机软件被普遍应用于各领域，快速变化的用户需求使得软件系统的规模和复杂性不断增加，随之而来的是大量软件缺陷的出现。隐藏的种种缺陷如果没有被及时发现，很容易被人恶意利用，从而泄露隐私数据，威胁系统安全，影响正常使用，具有很大的危害性。因此开发人员会利用各种代码分析工具检查软件的源代码，以便及时发现不符合编码标准的代码，并修复潜在的软件缺陷。根据分析时是否执行源代码，可以将这些工具分为静态分析工具和动态分析工具两类。静态分析工具关注源代码的词法、语法、控制流和数据流等静态信息，并与预设的缺陷模式相匹配，定位可疑的代码片段。常用的静态分析工具有 SpotBugs^①、JArchitect^②和 Coverity^③等。而动态分析工具在真实或虚拟的处理器上跟踪程序的运行路径，检查状态模型，监听并捕获那些可能由代码缺陷引起的程序行为。先前的研究也开发了一些用于检查验证 Java 程序的动态分析工具，比如 JNuke[1] 和 Kieker[2]。和动态分析工具相比，静态分析工具不必获得程序运行的动态信息，分析速度也更快。并且静态分析技术使得工具能够在软件开发的早期，也就是编码阶段，就能及时地发现代码中存在的各种缺陷，从而降低修复成本。例如静态分析工具 Findbugs 能识别 300 多种程序错误和可疑的用法，已被 Google 用于开发 Java 库的代码测试和审查流程，取得了不错的效果 [3]。

然而，静态代码分析工具在面对复杂多变的大型软件系统时，分析的效果往往依赖于其预置的缺陷模式。这些缺陷模式通常是由领域专家在专业知识和历史经验的基础上总结得来，需要对程序的执行状态做出一定的抽象，所以并不会非常精确。而且分析工具需要在漏报与误报之间作出权衡，考虑到缺陷的危害性，一般来说，选择的策略是牺牲一定的准确率而尽可能地报告出更多的潜在缺陷。这意味着得到的警告列表存在着许多误报，事实上并不意味着有

^①<https://spotbugs.github.io/>

^②<https://www.jarchitect.com/>

^③<https://scan.coverity.com/>

真正的代码缺陷存在，使得静态代码分析工具的检测效果大打折扣 [4]。例如先前的研究中表明，平均有 90% 的结果被标记为误报并会在之后从待修复列表中移除 [5]。这些误报使得开发人员不得不重新检查所有警告，以确定其究竟是否代表着真正的缺陷。这一过程相当费时费力，并且需要付出额外的成本，与借助静态代码分析工具尽快发现缺陷的初衷背道而驰。因此如何排列检测所得到的警告以尽快处理其中的正报，就成为了一个十分关键且亟待解决的问题。

虽然警告的特征多种多样，但是通过回归分析等特征缩减方法 [6]，可以过滤掉预测能力较低的特征。在这些可用特征的基础上，减轻误报警告影响的方法大体上可分为分类和排序两种。分类方法是将被正误报识别视作二分类问题，在所有警告被分为正报和误报之后，只审核那些被分类为正报的警告，而忽略被分类为误报的警告。分类一般通过训练机器学习模型来完成，但是训练机器学习模型需要大规模的警告正误报数据集，而通过人工逐个审核历史警告的方法获得正误报标签的方法无疑是不现实的。有的研究采用启发式的方法判断警告是否为误报 [7]，将各版本代码按时间顺序排列，如果警告在之后的版本中消失且出现警告的文件仍然存在，则可以认为警告在之后的版本中被修复，说明该警告是一个正报，否则为误报。但是，这一方法的鲁棒性不足，如果选择的版本序列发生变化，有可能得出不一致的结论，并且得到的结果可能和开发人员的意见相反 [8]。而排序的方法不需要标记警告，一般通过选择部分特征作为排序指标或者进一步组合出复杂的统计模型 [9]，尽可能将真正的警告排在前面给审核人员审核。这样一来，审核人员先检查的警告中大部分都是真正的缺陷，然后按顺序向后检查直到误报率增加达到某个阈值再停止。排序方法的缺点之一是，大多数研究中的排序结果是一次性的，排序完之后就不再变化。正如 Kremenek 等人 [10] 所说，静态的排序方法忽略了测试人员在审核过程中的反馈，考虑到警告之间可能存在的关联性，如果在审核时允许动态地对警告重新排序，可能进一步提高排序的质量。另外，分类或排序时会使用之前项目的历史数据，但是在新项目上进行预测时，由于新项目的代码特征和开发流程并不相同，基于历史数据的模型的有效性可能会表现得不如预期 [11]。

为了解决上述问题，本文提出了基于主动学习的警告排序技术，开发了静态代码分析工具警告排序系统。在软件开发阶段的版本迭代过程中，系统会使用静态代码分析工具扫描开发人员上传的当前版本源代码和 Jar 包，得到当前版本的警告列表并完成特征提取的工作。接着，系统利用主动学习技术的查询策略，在每次模型训练后动态地为没有审核过的警告排序，将排在最前面的部

分警告交由人工审核。审核人员将标记结果返回系统之后，系统将其作为已知标签的数据样本加入模型训练集，避免需要为模型提前准备大规模已标记数据集的问题。由于当前版本的警告数据和已训练的模型可以继续用在下一个版本的训练流程中，实现在同一个项目上持续改进模型的预测效果，而不是在不同项目上预测结果，因此不必担心跨项目带来的问题。最后，当反馈的正报数量达到相应阈值或模型对警告的分类结果不再变化时，自动结束主动学习流程。此时，审核人员还没有检查完所有警告，就能根据模型动态的排序结果，从众多警告中审核完大多数的正报，以此减轻审核人员的工作负担。

1.2 国内外研究现状

1.2.1 静态代码分析工具

近年来，静态代码分析工具被软件开发人员广泛地用于检测代码中隐藏的缺陷，以便在软件系统的开发阶段尽早地修复它们，因为在编码阶段更早、更多地发现缺陷有助于减少修复的成本 [12]。和依赖程序运行时行为的动态分析技术相比，静态分析技术可以抽象地搜索所有可能的程序行为和到达的状态，不受测试用例质量的限制。静态分析工具一般使用缺陷模式这一概念查找缺陷。缺陷模式是指使用语言特性或库 API 时不正常的用法，可以被简单的静态分析技术查找出来，并能扩展应用于复杂的程序。以 FindBugs 为例，其中包含了大约 50 种由 BCEL^①实现的缺陷模式检测器。这些检测器使用了访问者设计模式，在访问所分析应用程序的每个类时，执行特定的分析策略，包括 [13]：

- 类结构和继承层次：只分析类之间的层次结构关系而不分析其中的代码。
- 线性代码扫描：线性扫描分析类中方法的字节码，使用访问指令驱动状态机，这时并没有利用完整的控制流信息。
- 控制流：构造并使用准确的控制流图进行分析。
- 数据流：同时考虑数据流和控制流，比如空指针检测器。

Ferdian 等人 [14] 使用三种静态分析工具 FindBugs, PMD 和 Jlint 扫描了 Lucene, Rhino 和 AspectJ 这三个 Java 项目，分析得出了 200 多个缺陷，发现一些警告族内的多数缺陷都能被工具检测出来。以及 Marcilio 等人 [15] 关于 SonarQube 在具体开发中使用情况的调查也表明，静态分析工具的使用能改进

^①<http://jakarta.apache.org/bcel/>

整体的软件质量，帮助团队中的开发人员评估是否接受他人贡献的源代码。但是也正是由于静态分析只依赖于匹配特定的缺陷模式而并不会运行代码逻辑，因此这些警告不一定意味着真实缺陷的存在。尽管静态分析工具可以通过设置过滤文件等配置项^①来减少不正确的警告，但是静态分析工具的检测结果中仍然存在大量的误报，影响静态代码分析工具的实际使用效果，导致开发人员不愿意使用静态分析工具。Vassallo 等人 [16] 采访了 11 位使用静态分析工具的行业专家，了解他们是如何在不同的情况下使用静态分析工具，同时研究了工具是如何被应用于开源项目的流程中。他们了解到，使用静态分析工具仍不是一种普遍的做法，而且大多数使用工具的开发人员会关注特定的警告类别，并根据团队要求等因素有选择地优先修复特定的警告。这说明虽然不少开发者认识到了静态分析工具的潜力，但是需要更好的优先级选择策略来增强开发者的信心并进一步推广工具的使用。

1.2.2 警告误报检测技术

近年来，静态分析技术本身也在进一步发展，采用数学约束求解工具进行路径约减或可达性分析以减少误报率，或是寻找更为准确的缺陷模式 [17]。但也可以选择优化静态分析工具的检测结果，而不是直接改进静态分析方法，从扫描项目代码所得到的警告中区分正确的警告和虚假的警告，从这一角度出发减轻人工审核的负担。在主要的误报检测技术中，一种是利用机器学习技术尽可能地区分警告中的正报和误报，再分别处理；另一种是利用排序的思想将正报尽可能地排在前面，优先交由人工审核。

基于机器学习的误报检测技术

现有投入商业使用的静态分析工具尽管可以检测缺陷，但是会为了保证性能在可接受的范围内而使用不完善不健全的方法，在分析时牺牲一定的精度和稳定性。由于分析方法的固有缺陷不可避免，因此对于一些特定的代码结构，静态分析工具可能更容易将其作为警告报告，尽管它并不能算是一个真正的缺陷。基于这一假设，误报以及产生误报的代码之间，很有可能具有相似的特征，可以被区分和预测，因此可以使用机器学习技术来捕捉其中隐藏的特征。

Hanam[18] 等人在调查静态分析工具产生的误报中发现，许多被静态分析工具报告的警告其实是误报，是因为它们的代码具有相似的模式。而想要改进工具的分析结果，不论是对于工具的使用者还是开发者来说，手动查找并总结

^①<http://findbugs.sourceforge.net/manual/filter.html>

这些误报的代码模式是一件非常耗时的事情。他们根据开发人员的审核结果或者历史版本的推断为之前的警告标记类别，使用警告前后的源代码特征训练决策树、朴素贝叶斯和贝叶斯网络三种机器学习模型，最后使用模型挑选出更有可能是误报的警告。和之前的研究相比，他们的方法能识别出更多的正报，并且能进一步用在跨项目的检测中。

Heo 等人 [19] 利用了异常检测机器学习模型中的单分类支持向量机（OC-SVM），来捕捉没有缺陷的源代码的共同特征。这是一种用于异常检测的无监督学习算法，能够识别出与训练样本分布不同的异常数据。在实际应用之中，误报的数据量远远大于正报的数据量，数据集呈现不平衡的特点。因此可以将误报数据作为训练样本，如果新数据被分类为与训练数据相似则为误报，否则为正报。而且从直觉上来看，不包含缺陷的程序代码更有可能具有共同或相似的属性，而包含缺陷的代码是非典型的，由多种特征共同决定，因此难以表征。他们的方法成功改进了两种静态代码分析工具的检测结果，在保持原有精度的同时降低了它们的误报率，且能应对不平衡训练数据集的情况。

在使用机器学习模型的基础上，Shivaji 等人 [20] 提出了一种丢弃最低信息增益比特特征的特征选择方法。作为机器学习标准流程的一部分，特征选择方法不仅避免了无用的特征对分类器准确性的影响，而且减少了处理大型特征数据集所需要的时间开销。他们选择增益比作为确定最有效特征的指标，与信息增益相比提供了对分类结果贡献度的标准化度量。最终他们选择了原特征集合中大约 10% 的特征用于机器学习模型，并且在使用朴素贝叶斯模型时可以获得 0.96 的平均 F-measure 评分，表明预测得非常精准，避免了误报过多的问题。

虽然这些研究中，机器学习模型的应用在一定程度上达到了误报检测的预期效果，但是模型的实际效果不可避免地依赖于用于训练的数据集。在之前的实验中，有的数据集是研究人员专门编写的代码文件，其中一部分含有人工设计的缺陷，在使用静态分析工具扫描后，将结果与文件是否含有缺陷相对照，得到正报和误报的标记。但是这些数据集通常和实际工程中遇到的缺陷差距较大，导致在实际应用中的效果并不像实验环境下那样理想。而基于实际项目的数据集通常又数量有限，并且距今已有一段时间，随着编程语言的更新，其代表的缺陷模式可能已经过时。本文希望引入主动学习技术，使用审核人员及时的审核反馈结果作为训练数据集，不必提前准备大批量的训练数据。随着项目开发进度的推进，主动学习技术可以做到持续改进，专注于利用当前项目的模式特征，从而避免直接应用机器学习模型所带来的一系列问题。

基于排序的误报检测技术

为了最大限度地减少大量误报的影响，静态代码分析工具自身通常会允许用户按照一定的原则输出排序后的报告，比如根据内置的警告类型优先级，或者只是简单地根据类别、模式、类或包的字母顺序排列。这种排序方式非常粗糙，并不能有效地将正报排在警告列表的前面。

Shen 等人 [21] 提出了经过改进的漏洞检测工具 EFindBugs，使用有效错误警告排序策略。这一策略分为两个阶段，第一阶段为每个警告计算真实漏洞的可能性，并将可能性更高的警告排在前面，第二阶段则根据用户对错误警告的分类重新计算真实漏洞的可能性，并将排序更新为更为准确的结果。这样用户可以节约排除误报的时间，并根据需求判断警告是否值得被进一步检查。他们使用这一工具在大型 Java 项目上做了实证研究，和原始的排序方法相比，显著提高了准确率和召回率。

Liang 等人 [22] 提出了一种基于“通用缺陷相关行”的自动化方法来计算有效的警告优先级，开发并发布了一个在线代码缺陷分析服务 CODAS。他们使用基于自然语言处理的技术识别缺陷相关的版本，并在此基础上使用新的算法准确地识别“通用缺陷相关行”，最后选择三类影响因素作为警告的属性。他们还提出了一种新的启发式方法来标记警告的正确，如果警告在以后的版本中消失并在缺陷修复修订期间被修复，则警告被标记为正确。

Allier 等人 [23] 比较了六种不同的警告排序方法，其中三种是对单个警告进行排序，另外三种是按照警告对应的漏洞规则进行排序。他们发现对警告的排序效果比按漏洞规则排序的效果更好，说明漏洞规则本身并无优先级的高低之分，而是主要取决于代码的上下文，但并未对上下文的影响作进一步的探究。总体上来说，他们认为选择能适用于单个警告的排序方法更好，但是对于没有历史信息的新系统来说，按照规则排序的方法更容易使用。

主动学习模型中的查询策略部分也需要对剩余未标记的警告做排序，挑选其中一部分向审核人员分批次请求对警告的标记结果。而整个过程中警告审核的先后顺序则可看作是对所有警告的排序结果。由此看来，主动学习本身就 and 排序方法有相似之处。另外，在没有历史版本的新系统上，主动学习技术也可以通过初始化采样从零开始构建训练集，解决了上文 Allier 等人提到的问题。

1.2.3 主动学习技术

现有的机器学习模型需要大量经过标记的数据作为训练集，而人工审核是一件既费时又费力的工作。目前主动学习技术在软件工程的多个研究领域都有应用，例如恶意代码检测 [24]，样本平衡 [25] 等方面，提供了一种减少人工审核成本的思路，同时还能获得良好的性能。

Zhe Yu 等人 [26] 研究了主动学习技术在解决软件工程中总召回问题上的应用，并描述了应用主动学习的一般流程。总召回问题的任务是检查和标记尽可能少的候选样本，同时达到比较高的召回率。由此可见警告优化也属于总召回问题，即在审核尽可能少的警告的情况下，找出大多数误报以达到较高的召回率。他们也指出，需要进一步改进核心算法以更少的成本实现同样的召回率，或者构造更准确的预测模型。

Zhe Yu 等人 [27] 还提出了 HARMLESS，一种基于增量支持向量机和主动学习技术的工具，依据已经检查审核过的源代码构建漏洞预测模型，并指出下一步应当审核的源代码文件。通过这种方式，HARMLESS 可以提示审核人员接下来应当检查的可疑源代码，以减少发现漏洞所需要付出的时间和精力，进而保护软件免受过多安全漏洞的影响。

Junjie Wang 等人 [28] 提出了基于局部性的主动学习分类方法（LOAF）来对众包测试报告中真正的缺陷进行分类。他们认为之前的主动学习技术隐藏着数据同分布的假设，但是众包测试报告却具有显著的局部性偏差，相同的特征可能只由一部分报告共享。因此他们实现的 LOAL 会在样本的局部邻域内寻找信息量最大的数据样本，报告给审核人员标记，然后基于局部邻域的数据训练分类器。在他们的实验中，LOAF 的性能甚至优于传统的基于大量已标记的历史数据的监督学习方法。

由此可见，主动学习技术的特点在于无需使用大规模的已标记数据集，从而降低数据收集和训练上的成本开销。Yang 等人 [29] 的研究说明，对于一些机器学习模型，例如支持向量机，使用少量的数据实例训练也可以获得有效的模型。在对警告分类的场景下，对数据实例的标记具有高昂的成本，主动学习可以通过优先选择高信息量的实例来减少此问题带来的影响。

1.3 本文主要工作

针对以上提到的研究现状和当前警告误报检测工作所遇到的问题，依据对警告列表进行进一步排序的思想，本文提出了一种基于主动学习技术的警告排序方法，并在此基础上实现了警告排序与审核系统。本系统能够在软件开发的过程中，对每个版本的代码自动执行静态代码分析，结合主动学习流程优先将最有可能是真实缺陷的警告交给审核人员审核。

在项目的编码开发阶段，完成一个版本的迭代开发之后，管理员将源代码打包提交给警告排序优化系统。系统将自动使用静态代码分析工具扫描项目生成扫描报告，接着基于位置、片段或哈希值过滤掉之前版本中已出现的警告，然后启动主动学习流程。在主动学习的过程中，系统提取警告自身特征和分布特征、相关源代码的度量特征和分析特征，然后据此训练机器学习模型。之后系统会对没有经过人工审核的警告集合进行排序，依次选择最有可能是正报的警告交给审核人员审核。

审核人员会收到系统分发的审核任务，并阅读相应的源代码，以此判断其是否为一个真实的漏洞，并将判断结果反馈给系统。系统收到判断结果后利用已知的数据集继续训练模型，并使用模型进行预测，以此判断最有可能正确的警告，再次交由人工审核。系统将重复循环直到满足退出循环的条件。

最后我们对系统进行了测试，保证了系统的可用性和可靠性，并通过设计的实验验证了我们基于主动学习的误报检测方法的有效性，说明系统可以满足预先设定的业务需求，具有一定的使用价值。

1.4 本文组织结构

本文将通过以下六个章节详细介绍提出的基于主动学习的警告排序技术，以及在此基础上构建的警告排序系统。

第一章，引言。本章首先介绍了静态代码分析工具以及目前工具存在较高误报率的问题，分析了现有的基于各种警告特征的警告结果优化方法，包括分类和排序。然后介绍了警告误报检测和主动学习技术的研究现状。最后在此基础上，说明了本文的主要工作内容和组织结构。

第二章，相关技术综述。本章介绍了系统中使用的主要技术，包括警告特征提取、主动学习、集成学习和前后端框架等。

第三章，警告排序系统的需求分析与设计。本章介绍了系统的需求分析、概要设计、详细设计，采用系统顺序图、流程图、UML 类图对相关模块的具体设计进行了论述。

第四章，警告排序系统的实现。本章详细描述了系统项目管理模块、特征提取模块、主动学习模块和警告审核模块的代码实现并给出了相关核心代码。

第五章，实验设计与系统测试。为了证明本文所提出的基于主动学习的警告排序方法的有效性，本章介绍了相关实验的设计、配置与结果，和进行系统测试的环境、过程和最后的测试结果。

第六章，总结和展望。对本文所完成的工作进行总结与说明，分析了目前系统的不足之处，针对性地提出了可能的改进方向。与此同时还对未来警告排序技术发展方向做出了展望。

第二章 相关技术综述

2.1 主动学习

在众多机器学习算法中，最为常见的一类是有监督学习，即用于训练的数据有确定的分类标签，算法相当于为数据的特征与标签之间的关联性建立模型，从而预测未知数据的类别。正因如此，分类标签的准确度极大地影响着模型的预测效果。在实际应用中，存在很多需要人工标注数据标签的场景，例如在典型的图片分类任务中，需要手动为每张图片打上标记，标出所识别物体的位置和类别。在专业性较强的应用场景下，例如识别医学成像数据辅助决策，甚至需要专家的专业知识为未知分类的图像标记类别。可见获得高质量的数据分类标签有时要耗费很多时间和精力，其高昂的成本令人难以接受。

而警告的正误报分类也是件专业性较强的工作，需要具备软件安全相关知识的审核人员才可完成。为了降低正误报标记所带来的成本，本系统引入了主动学习技术，通过结合主动学习的查询策略和排序优化的思想尽快审核系统中真实的漏洞。主动学习技术能够在仅有少量标记数据的数据集上进行训练，再从当前未标记的数据集合中主动选择下一步要人工审核标记的数据。这一过程大大减少来训练模型所需要的标记数据数量，同时还可以获得更快的训练速度。在理想的情况下，随着已标记数据数量的增加，模型的性能也会随之提高，并逐渐达到一个稳定的水平。

2.1.1 场景分类

总体上来说，主动学习的模型场景可以分为以下三种 [30]:

- 成员查询
- 基于流的主动学习
- 基于池的主动学习

成员查询（Membership Query）是指在主动学习过程中，学习器自行构造属于输入空间的一个未标记的数据样本向人工询问其是否属于某一集合作为

标签，而不是从潜在的自然分布中采集数据样本 [31]。在数据空间有限的情况下，成员查询通常比较容易处理，并且效率也比较高。如果主动学习系统难以构造相关的样本，这时构造子集或超集的查询可能更为有效。而在数据空间几乎无限的情况下，例如手写字符识别或语音识别，学习器可能只能创建出杂乱无章的图片或语音流，很难合成出有意义的数据向人类查询标签。

基于流（Stream-Based）的主动学习每次从数据源中提取一个样本，由主动学习模型决定是查询它的标签还是丢弃它 [32]。这一方法基于未标记样本易于获得的假设，直接从实际分布中采集样本。如果输入分布比较均匀，那么看起来可能和上文的成员查询比较类似。可以通过计算样本的信息量阈值判断是否查询提取的样本，或者判断其是否在计算得到的不确定区域中。这种方法有时也被称为选择性抽样，但是一些文献中也会指代下文基于池抽样的场景。

而在许多现实生活的问题中，人们可以一次性地收集大量未标记的数据，这种场景下可以使用基于池（Pool-Based）的主动学习技术 [33]。这一方法假设在一个固定的数据池中有一小部分的已标记数据，剩下的都是未标记数据。当查询选择数据时，是根据信息量或其他度量指标对池中所有未标记的实例进行评估，或者如果未标记的数据量过大，则只选择其中的部分样本进行评估。与基于流的主动学习技术相比，前者是按顺序扫描数据并对每个数据独立地做出查询决策，而后者在选择最佳的查询实例前会对整个集合进行评估和排序。通常基于池的方法更为常见，但是当内存或处理器性能受到限制时，基于流的方法可能更为合适。

在本文的应用场景中，警告由静态代码分析工具在输出扫描报告时一次性生成，在查询之前为剩余的所有警告排序，因此属于基于池的主动学习场景。在这一场景中，初始状态下存在一个没有经过标记的数据集 E ，和已标记数据集 $L = \emptyset$ 。此时主动学习的流程如下：

1. 使用随机采样等初始化采样方法，从 E 中选择要标记的样本 x ，加入已标记数据集中，即 $L = L \cup x$ ；
2. 重复步骤 1 直到已标记数据集中的正样本数目达到阈值， $|L|_{positive} \geq R$ ；
3. 使用已标记数据集 L 训练或更新内部的机器学习模型；
4. 使用已训练的模型预测未标记数据集 $E \setminus L$ 内的样本分类，并选择新的需要标记的样本 x ，加入已标记数据集中 $L = L \cup x$ ；
5. 重复步骤 3 和步骤 4 直到符合停止条件。

2.1.2 查询策略

不论实例是主动学习模型自己生成的还是从给定分布中采样的，以上的主动学习场景都涉及到计算评估未标记实例的信息量。到目前为止，已有许多研究提出了不同的查询策略的方法。

最简单和最常用的查询方法是不确定性采样 [33]，它的思想是主动学习模型查询那些最不确定如何标记的实例，以此改进模型的效果。例如对于输出二分类概率的模型，不确定性采样只需简单地查询那些输出概率最接近 0.5 的实例。考虑更为普遍的一般情况，通常会采用信息论中表示分布所需信息量的熵作为不确定性的度量 [34]，这很容易推广到概率输出的多标签分类器以及更复杂结构化模型的情况。也有的研究采用其他指标，例如置信度作出不确定性的度量 [35]。对于部分复杂的模型，也有研究给出了其他方法，例如基于最近邻的分类器可以借助每个邻居对类标签的投票比例作为标签概率，基于支持向量机的分类器可以选择查询最接近线性决策边界的实例。

另一种查询策略是基于投票的查询算法 [36]，其中包含了一系列在同一标记数据集上训练的模型，每次选择查询样本时由各个模型对候选样本的标签进行投票，信息最丰富的查询即为他们最不同意的样本。如果我们将机器学习模型训练视作在样本空间内搜索最佳模型，那么主动学习使用尽可能少的标记样本，可以很好地达到限制样本空间的大小的目标。可以使用投票策略的前提是构建代表不同区域的模型，以及模型之间能存在一定程度的分歧。

最后一种查询策略是选择那些将使得当前模型产生最大变化的样本。其中一种由 Settles 等人 [37] 实现的方法是计算预期梯度长度，即计算当样本加入训练集后，目标函数的梯度将产生的变化。这一策略的前提假设是，去优先选择那些对模型影响最大的实例，而不管其查询后的标签如何。在之前的实证研究中，这种策略也能取得很好的效果，但是如果特征空间和标签的种类集合都非常大，则计算量会十分庞大。

其他查询策略尝试使用了其他指标，例如减少的方差 [38]、Fisher 信息比 [39]，和上述策略均有不同程度的相似之处。

2.2 集成学习

集成学习是机器学习领域下的一类通用方法，简单地说，它通过组合来自多个模型的预测结果以取得更好的预测性能。集成学习基于“群体智慧”的假

设，也就是一群对某个主题具备一般知识的人可以为诸如数量预测、空间推理等问题提供相对准确的答案。将一群人给出的答案综合到一起之后，结果中的噪音有可能相互抵消，使得结果会优于领域专家的结果。同样的原理也适用于依赖机器学习的人工智能程序，对于许多问题，集成学习获得的结果可能比组内任何一个单个成员的结果更准确。

集成学习通常由一系列“弱学习器”组成，因为即使经过大量的配置和调整，它们也很难达到所需的预测准确度。集成学习通过组合这一系列“弱学习器”的结果，得到的汇总结果可能会优于原本任何一者的预测结果。集成学习之所以有效，是因为每个独立模型可能会陷入局部最优解，使得在其中的部分数据上表现良好，而在其他数据上则不太准确。通过组合多个学习器的结果，集成学习降低了获得局部最小值的风险，扩展搜索空间从而更好地适应数据空间，这使得它们的弱点有可能得到抵消。集成学习有多种组合模型的方法，以下三种方法最为常见：Bagging、Boosting 和 Stacking。

2.2.1 Bagging 方法

Bagging 方法是一种通过有放回抽样在多个数据集上生成不同预测器，并使用它们获得一个聚合的预测器的方法 [40]。构建数据集的方法引入了随机化的过程，通过随机抽样的方法在原始训练集上构建多个随机子集，并分别作为每个预测器的训练集。聚合的方法包括在预测时对标签进行投票或者计算预测概率的平均值，两种方法通常分别被用于分类和回归。如果使用的预测器容易受到训练集的干扰，那么 Bagging 方法将显得很有效，提高准确率。在许多情况下，Bagging 方法无需调整底层的基本算法就可以方便简单地改进单个模型，减少模型的过度拟合。

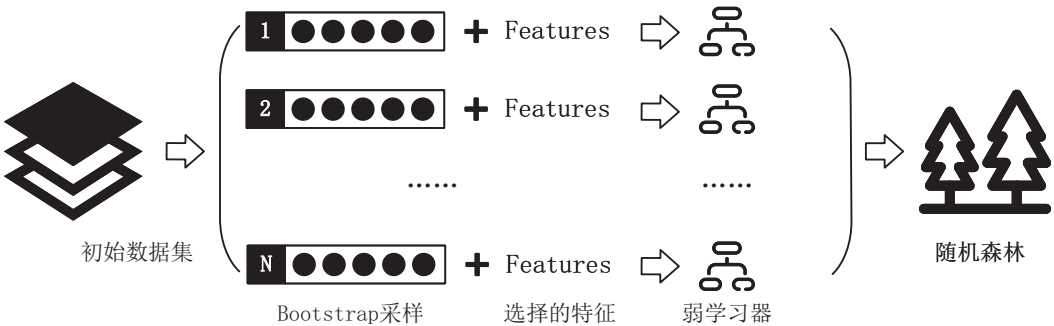


图 2.1: Bagging 方法示意图

例如图 2.1 中随机森林就属于 Bagging 方法的一种，是用随机的方式建立由

很多的决策树组成的森林，随机森林的每一棵决策树之间独立且没有关联。在随机森林中，同时进行两种随机采样的过程：对输入数据表格的行，也就是数据的数量，和数据的列，也就是数据的特征，都进行采样。对数据的行进行采样时采用有放回的方式，对数据特征则选择出数量远小于总特征数的子集。预测的时候，随机森林中的每一棵树的都对输入进行预测，最后进行投票，获得票数多的类别作为最后分类的结果。

2.2.2 Boosting 方法

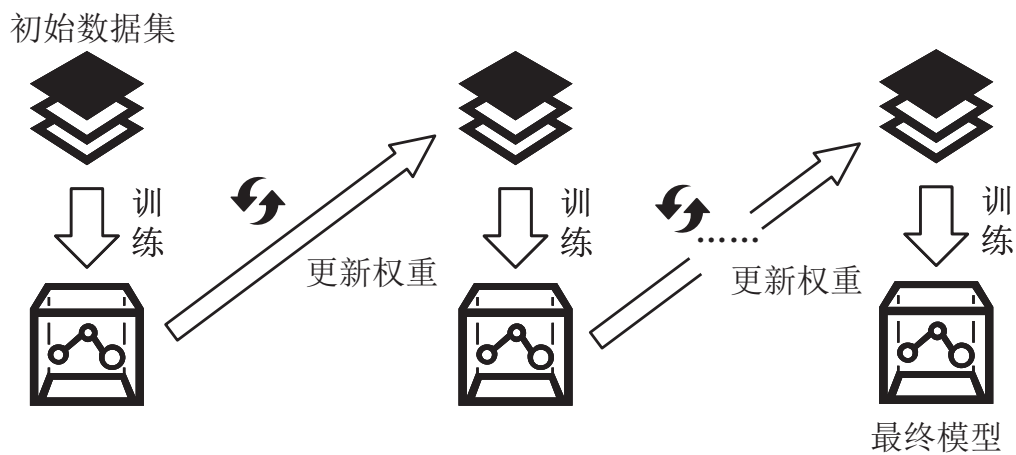


图 2.2: Boosting 方法示意图

Boosting 中有代表性的是 AdaBoost (Adaptive Boosting) 算法 [41]，如图 2.2 所示是在同样的数据集上不断训练模型，起初每个样本的权重是相同的，但每次训练后会根据预测结果更改样本的权重，下一次训练时模型会利用经过加权的数据集学习。样本的权重会根据预测的结果更改，一般来说，被正确预测的样本权重会被削弱，而错误的样本权重会增加。随着迭代的进行，难以被准确预测的样本权重会越来越大，即对模型产生更多的影响。重复这个迭代过程，最终得到一系列模型，由此可以构建一个由学习器性能进行加权计算的线性组合，得到最终结果。

2.2.3 Stacking 方法

Stacking 方法是一种组合学习器以减少其误差的方法，每个学习器的预测输出作为最终学习器的输出 [42]，如图 2.3 所示。在原始的训练数据集上通过随

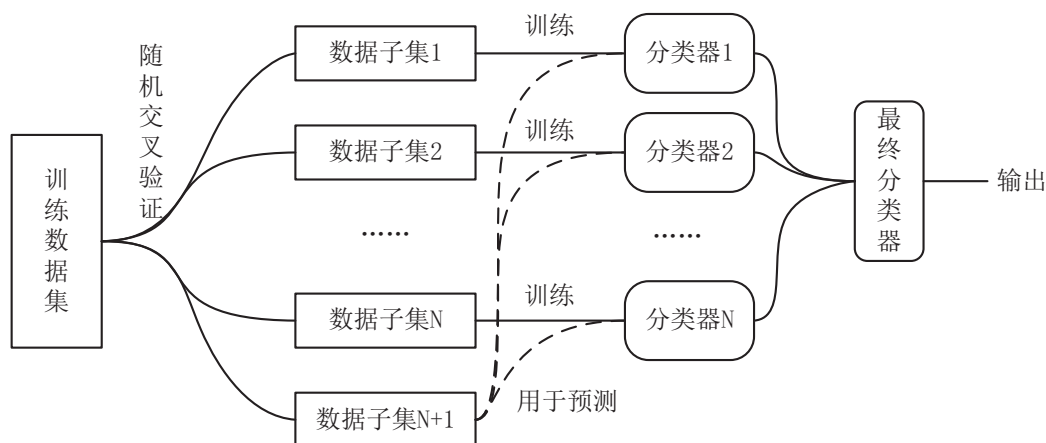


图 2.3: Stacking 方法示意图

机交叉验证的方式得到一系列数据子集，使用其中的 N 个子集分别作为训练集训练 N 个弱分类器，并分别对另一个独立的子集做出预测，这样训练集和预测集相互独立。 N 个弱分类器的结果会作为最终分类器的输入，最后得到的输出即为集成学习整体的预测结果。在实际应用中，通常使用 logistic 回归作为最终学习器的组合策略。

2.3 警告特征提取

在目前的警告优化思路之中，不论是分类方法还是排序方法，警告特征都是有效区分正报警告和误报警告的关键影响因素。而在不同的研究中，对特征的定义常常是模糊不清甚至相互矛盾。在不同的项目上评估特征的有效性时，有时也会得出相反的结论。为了解决这些使用警告特征时的问题，Junjie Wang 等人 [43] 的研究系统地调查了不同文献中提取警告的方法，将警告的特征划分为以下几类，并评估了警告在实验中的有效性。

- 文件特征：警告所在文件的特征；
- 文件历史：警告所在文件的变更历史特征；
- 代码特征：警告对应代码的静态特征；
- 代码历史：警告对应代码行的修改历史特征；
- 代码分析：警告对应代码的程序分析模式特征；
- 警告特征：警告自身的特征；
- 警告历史：警告自身的变更历史的特征；
- 警告组合：警告特征的组合以及其他信息。

Heckman 等人 [44] 的研究中将警告特征分为以下几组：

- 警告标识：静态代码分析工具在生成警告的同时也会生成警告的标识，一般来说都会包括以下内容：项目名称、包名称、文件名称、方法签名、警告类型、警告分类、优先级、描述信息、警告所在行号。
- 警告历史：从最初版本的源代码开始，生成每个版本代码的警告列表，主要通过确定警告的创建和关闭，测量其在历史上发生变化的次数。
- 代码度量：软件规模和复杂性的指标常常被用于衡量软件发生故障的可能性，可见其同样与报告警告的概率有关，可以作为警告的一类特征。在之前的研究中，使用 JavaNCSS 工具^①获取源代码的度量特征，包括包、类、方法的大小，文件与包中类或方法的数量，和代码的圈复杂度。
- 代码历史：将警告的历史与文件修改的历史相关联，可以用于确定警告创建和关闭发生在哪个版本 [45]。除此之外还可以获得文件被创建和删除的版本，文件和包级别最后一次修改的版本，和修订版本的开发者列表作为这一类的特征。
- 代码变更：源代码的变更历史记录项目随时间变化的过程，因此文件、包和项目级别的代码在不同版本之间的变化量也可以作为特征之一，这些变化会在代码被提交到版本控制系统（如 Git 或 SVN）时记录下来。
- 聚合特征：在上述特征的基础上做一些复合的计算可以得到更多的特征，例如版本创建和关闭的警告总数，警告创建和关闭间的存在时间等等。

Heckman 等人 [46] 还提出了一种通过构建特定于项目的机器学习模型的警告优化方法，收集静态分析生成警告的特征并使用特征子集评估算法来选择重要的特征集合。他们的实验结果表明，文件的相关特征、部分代码度量特征和文件变更历史特征，被认为是能够有效识别的特征子集。因此，本文在他们的结论基础上，提取了相似类型的特征来进行基于主动学习的警告排序优化。

2.4 Flask 框架

Flask^②是 Python 的一个微型轻量的 Web 框架，虽然只提供了基本的核心功能，但是在开发方面具备一定的可扩展性，能节约构建 Web 应用程序的时间。Flask 的核心组件包括 Jinja 模板引擎和 Werkzeug WSGI 工具包。Jinja 是一个功

^①<http://www.kclee.de/clemens/java/javancss/>

^②<https://flask.palletsprojects.com/>

能齐全的模板引擎，提供了诸如沙盒执行、XSS 预防、模板继承、易于调试、可配置的语法等功能和特性。其文件结构分为静态文件和模板文件两部分。模板文件包含所有 Jinja 模板，也就是 Html 页面。静态文件包含网站所需的所有静态代码，如 CSS 代码、JavaScript 代码和图像文件。此外，HTML 模板中编写的代码也能被编译为 python 代码。WSGI 应用程序框架提供了交互式调试器、完整的请求对象、端点路由系统、用于处理实体标签的 HTTP 工具、缓存控制、日期、cookie 等功能 [47]。它还为本地图开发提供了一个多线程的 WSGI 服务器，用于模拟并测试 HTTP 请求。

Flask 通常被称为原型框架，不包括数据库的抽象层或任何类型的安全验证等功能。因此，Flask 为实现者提供了充分的灵活性来添加想要的需求，例如：

- 用于数据库的 SQLAlchemy
- 用于表单验证的 Flask-WTF
- 用于缓存管理的 Redis
- 用于限制请求速率的 Flask-limiter

Flask 能自动处理请求上下文，因此不需要将请求对象显式地传递给函数。Flask 提供了请求上下文作为全局对象，可以在整个应用程序生命周期中都能访问到请求对象。对于浏览器客户端用户而言，路由是其访问的主要接口，因此在处理函数上使用装饰器修饰的 URL 模式匹配后，相应的处理函数就能够获取请求对象和主体。和另一个被广泛使用的 Django 框架^①相比，Flask 最适合原型设计和小型项目，而 Django 需要更多的学习成本且更适合开发大型项目。虽然在之后的管理和维护方面，使用 Flask 需要比使用 Django 付出更多的工作。

2.5 Vue.js 框架

Vue.js 是用于开发交互式 Web 界面的渐进式 JavaScript 框架，关注重点在前端的视图部分，与其他项目和库集成非常容易。换句话说，如果我们有一个使用其他 JS 框架开发的现有项目，我们可以轻松地使用 Vue.js 为现有的项目构建用户界面，其中 Vue 将处理 UI 的前端视图，而其他框架可用于服务器端渲染和数据处理等功能。

和其他框架（如 React、Ember 等）一样，Vue.js 也使用虚拟 DOM，即创建以 JavaScript 数据结构形式存在的 DOM 副本，避免直接对 DOM 进行更改。

^①<https://www.djangoproject.com/>

每次修改时会先更改数据结构，与原始的数据结构进行比较，最后才将更改更新为真实的 DOM 并对用户可见。虚拟 DOM 的使用优化了框架的性能，用户可以更快地看到更改后的结果。Vue 提供的数据绑定功能有助于操作或为 HTML 属性分配值、更改样式，以及在 `v-bind` 的绑定指令的帮助下分配类。Vue 的其他特性还包括：组件、路由、计算属性、观察者、模板等等。

由于其简单易用，比其他许多 JS 框架更灵活，以及相对平缓的学习曲线，Vue 在小型项目中越来越受欢迎。

2.6 RabbitMQ 消息队列

RabbitMQ^①是一个由 erlang 开发的应用高级队列协议（Advanced Message Queue Protocol, AMQP）的消息代理中间件，负责从“生产者”处接收消息，然后传递消息给“消费者”，过程中可根据规则路由、缓存、持久化消息。AMQP 是面向协议和消息的中间件服务的开放且标准的应用层 [48]，允许用户指定可以接收的消息，以及如何在安全性、可靠性和性能之间获得平衡。

消息队列允许 Web 服务器快速响应请求，而不是被迫等待执行那些耗时严重的过程以至于大大增加响应时间。尤其是当需要将消息分发给多个消费者或平衡消费者之间的负载时，消息队列也十分有用。当消费者从队列中取出一条消息并开始处理时，生产者正在生产新消息，两者的分离意味着消费者可以部署在与生产者完全不同的服务器上。甚至消息体可以用一种编程语言创建，并用另一种编程语言处理。两个应用程序只会通过它们相互发送的消息进行通信，这意味着发送者和接收者的耦合度很低。

2.7 本章小结

本章介绍了本文系统开发中涉及的主要技术和使用的主要框架。首先，我们介绍了主动学习，包括其不同场景下的实现和多种查询策略。其次，我们介绍了集成学习的几种主流的方法，包括 Bagging、Boosting 和 Stacking 方法，将作为主动学习流程中使用的机器学习模型。然后我们介绍了现有研究中不同的警告特征提取角度和具体的警告特征。最后，我们介绍了系统开发时用的后端框架 Flask、前端框架 Vue.js 和服务器异步执行任务的 RabbitMQ 消息队列。

^①<https://www.rabbitmq.com/tutorials/tutorial-one-python.html>

第三章 警告排序系统的需求分析与设计

3.1 系统整体概述

在基于主动学习的警告排序方法的基础上，本文构建了一个面向软件开发人员和测试人员的警告排序系统。本系统的工作目标是帮助开发人员尽快找到警告列表中真实存在的缺陷，同时尽可能减轻审核人员的工作负担，不必审核完所有警告就能找出大部分的正报警告。通过这一系统，开发人员可以在更短的时间内修复可疑的缺陷代码，提升软件的质量，而且可以节省审核人员人工审核的成本开销。本系统的总体流程如图 3.1 所示。

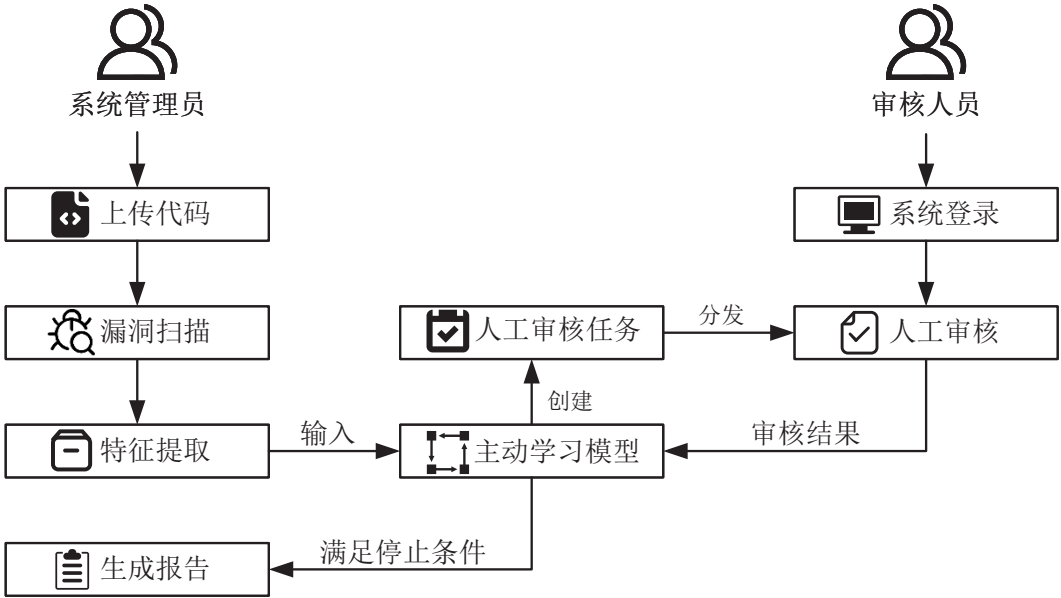


图 3.1: 系统总体流程图

本系统中的系统管理员一般指团队中特定的软件开发人员，具有相应管理权限，能够上传整个项目的源代码和编译打包好的 Jar 文件。在上传之后，系

系统将自动运行静态代码分析工具扫描项目获得警告，从多个方面提取警告的特征，并启动主动学习模型的训练流程。主动学习模型在不断迭代训练的过程中，会根据查询策略的需要创建人工审核任务，用来向审核人员询问数据样本的标签。系统会自动将人工审核任务分配给不同的审核人员，由于本系统并不涉及多用户的结果投票策略，因此人工审核任务通常只分配给一个审核人员完成。系统中的审核人员由具备一定专业知识的测试人员担任，登录系统后就会收到系统分发的审核任务，根据展示的源代码以及系统的提示完成审核并提交结果。主动学习模型会一直等待，直到收到这一批所有的审核结果后继续其循环流程，在满足设定的停止条件后退出循环并生成报告。最后系统管理员可以查看最终的警告报告列表，处理其中标记为正报的警告。

3.2 系统需求分析

系统分析设计的第一阶段是进行需求分析，主要目标是了解用户非形式化的需求，并将其转化为完整的需求定义，再转换为相应需求规格说明。本节首先确定了系统的关键涉众，描述了涉众的特征以及他们在用户视角下对系统的期望。之后从系统的角度理解软件，定义了功能性需求和非功能性需求。最后分析系统各个用例，构建用例模型，包括用例规格说明和用例图。

3.2.1 涉众分析

本系统作为软件开发团队内部使用的警告审核平台，仅包含系统管理员和审核人员两个角色，其涉众分析如表 3.1 所示。

系统管理员作为系统的管理方，可以通过系统上传项目源代码，运行静态代码分析工具，调用主动学习模型，管理审核人员和查看最终的审核报告。他们很大可能也是开发团队的管理者，拥有一定的权限和管理能力，根据最后的审核报告对源代码进行复查和缺陷修复。他们希望系统能尽可能提高正报识别的准确率，以减少他们二次审核的负担。他们对于系统的开发呈现积极的态度，希望系统能够尽早上线以支持他们的工作。

审核人员作为系统最大的使用群体，是系统发布的审核任务的完成者，可以通过系统查看扫描工具得到的警告详情和相应的源代码片段，以及最为关键的填写审核结果。他们在团队中一般扮演测试人员的角色，具备软件缺陷的相关专业知识，能快速阅读源代码并做出判断。他们希望系统能为警告审核的工

表 3.1: 涉众分析表

涉众名称	涉众特征与期望
系统管理员	作为系统的管理方，他们希望在软件项目的一个版本开发完毕之后，能在系统的帮助下，较为准确地获取可疑的缺陷代码列表。系统管理员在软件开发小组中应当具有一定的权限，能访问仓库的完整代码并编译打包成成品。同时应当具有一定的管理能力，能管理已在系统中注册的审核人员。
审核人员	作为系统审核任务的完成方，他们希望能在排序优化系统的帮助下，方便地阅读警告的相关信息，包括漏洞扫描工具的检测结果和源代码文件中对应的代码片段。审核人员应当具备系统安全方面的知识，能快速阅读源代码并较为准确地识别缺陷，并将自己的判断结果返回给系统。如果有可能的话，他们还希望系统能尽可能地减轻他们审核的负担。

作提供比较好的用户体验，提高审核的效率。他们比较愿意通过系统完成审核任务，但也担心系统不够智能，分发过多的任务给他们审核。

3.2.2 功能性需求

功能性需求的构建涉及对组件及其环境的相关内部状态和行为进行建模，这一概念主要与使用环境有关，而和设计或实现没有必然联系。功能性需求有助于设计人员之间以及与用户之间的相互交流 [49]。在本文的系统内，其行为可以用下面这一完整的流程来描述：

1. 开发团队内的测试人员在系统内注册成为审核人员，项目的管理者拥有管理员账号和密码。
2. 项目的一个版本开发完成时，管理员上传项目的源代码，编译打包后的可执行文件，并填写相关信息，提交给系统创建警告扫描任务。
3. 系统完成警告扫描、特征提取等一系列操作后，在主动学习模型的执行过程中会创建审核任务，用以向审核人员询问审核结果。
4. 审核人员登录系统后查看待审核的警告列表，并逐个阅读警告的细节和代码，利用自己的专业知识判断其是否为一处真正的缺陷，并及时地将结果反馈给系统。
5. 主动学习流程满足停止条件后，系统管理员可以查看最终的报告，包含原始的警告列表以及系统和审核人员对其是否为正报警告的判断。

经过进一步的识别和整理，本文使用表 3.2所示的功能性需求列表来描述所设计系统的功能性需求。

表 3.2: 系统功能性需求列表

ID	需求名称	需求描述
R1	查看审核任务列表	用户可以查看系统为其分配的审核任务。用户可以看到审核任务所属的项目名称、项目版本、发布时间等信息。在用户处理已有审核任务的同时，系统可能会为其分配新的任务，用户可以通过刷新页面查看。
R2	查看警告信息	用户可以查看审核任务中详细的警告信息，包括警告的优先级、等级、类型、代码行，和对应源代码所在的文件路径、类名、方法名、方法签名等信息，并展示代码片段。同时系统还会给出当前模型对于警告为正报的预测概率，以辅助用户判断审核。
R3	填写审核结果	用户可以在查看警告信息后填写审核结果，标记警告是否为一个正报，并保存审核结果。
R4	账户注册	用户可以在系统输入用户名、邮箱和登录密码以创建审核人员账户。
R5	查看项目列表	管理员可以查看系统中存在的项目列表。管理员可以看到项目的名称、版本、上传时间、标记完成时间、当前状态等信息。
R6	上传项目	管理员可以上传项目的源代码，和打包后的 Jar 包，并填写其他信息。填写完毕后管理员上传项目，系统将自动开始执行后续的流程。
R7	查看最终报告	管理员可以选择一个已经全部标记完成的项目版本，查看其最终的报告。管理员不仅可以看到静态代码分析工具给出的警告信息，还可以看到审核人员的审核结果。
R8	审核人员管理	管理员可以查看系统内已注册的所有审核人员名单，以及他们的标记审核情况。

3.2.3 非功能性需求

非功能性需求不包括对系统特定行为的需求，而是偏向于描述为满足业务需求需要达到的软件系统质量属性。非功能性需求的存在确保了良好的用户体验并简化了软件的操作，保障了软件系统的可靠性，可用性和性能。下文从性

能需求和质量属性两方面描述了系统的非功能性需求。

系统的性能需求包括：速度、容量和负载：

- 速度：对于所有立即返回结果的用户请求，系统必须在 1.5s 内返回响应。
- 容量：系统能够存储 100 个版本的项目以及他们的扫描报告、特征数据集和审核结果。
- 负载：系统允许最多 50 人同时访问并保证处理请求的速度没有明显变化。

系统常见的质量属性包括：易用性、安全性、可靠性、可用性、可移植性和可维护性 [50]：

- 易用性：易用性与用户完成期望任务的难易程度有关，体现了软件系统易于被使用的程度。本系统的主要使用者都是软件开发人员，应具有较强的易用性，减少在学习使用系统和人机操作上耗费的时间。系统的界面应当简洁清晰，使用户能够在十分钟之内理解系统的操作流程。
- 安全性：安全性是指为了保护系统安全，在向合法用户正常提供服务的情况下，阻止未经授权访问的能力。系统中注册的审核人员账号只能访问其允许访问的相关功能，未登录的用户访问系统时将自动跳转至登录界面要求输入登录凭证。
- 可靠性：可靠性是指系统在一定时间内和规定条件下稳定运行的能力。系统在运行期间如遇网络故障，不能因程序错误中断出现错误的结果。
- 可用性：可用性要求系统做好错误处理工作，减少故障发生的概率。系统的可用性要求出现异常时能够及时的捕获并给出相应的提示。
- 可移植性：可移植性是指系统在不同的硬件或软件系统之间迁移的能力。系统主要由 Python 编写，其可移植性依赖于 Python 解释器的跨平台性。
- 可维护性：可维护性是指在出现意料外状况或故障时，能快速定位故障并解决问题使系统正常运行的能力。系统的长时间运行要求系统具备一定的可维护性，保持业务持续性的同时降低运营成本。因此系统需要设计有详细的日志记录功能。

3.2.4 用例模型描述

本系统可以划分成八个用例，系统用例图如图 3.2 所示。系统用户角色分为审核人员和系统管理员。审核人员的用例包括：查看审核任务列表、查看警告信息、填写审核结果和账户注册。管理员的用例包括查看项目列表、上传项

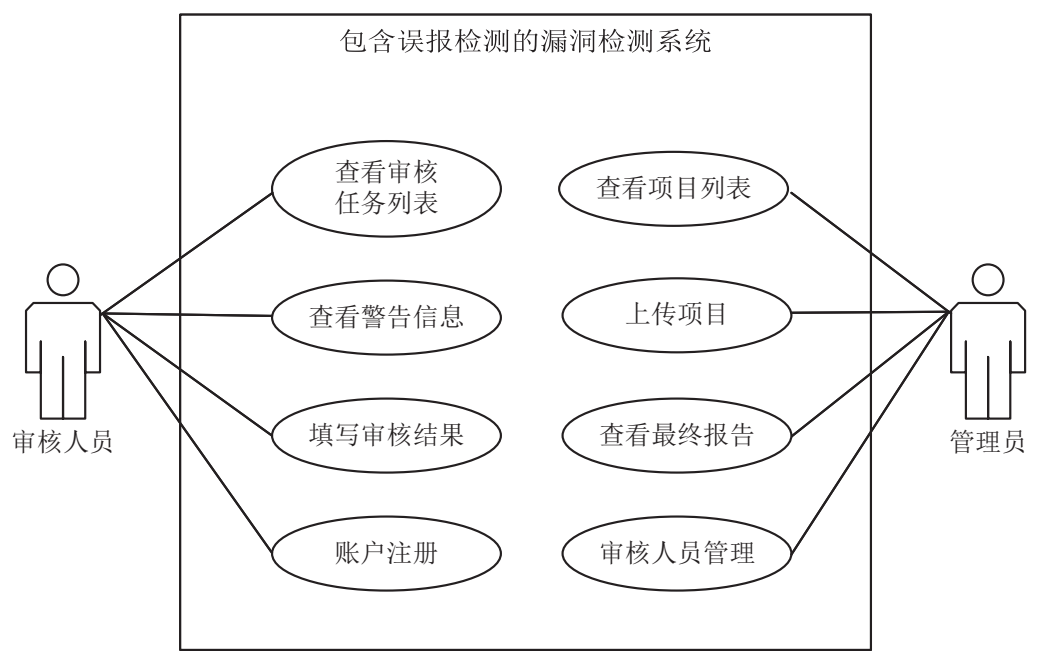


图 3.2: 系统用例图

目、查看最终报告和审核人员管理。

表 3.3: 查看审核任务列表用例描述

ID	UC1
参与者	审核人员
触发条件	审核人员进入系统首页
前置条件	审核人员登录系统并通过认证
后置条件	系统提供分配给审核人员的所有审核任务列表
优先级	高
正常流程	1. 审核人员登录后自动跳转至首页，或点击“任务列表”回到首页； 2. 系统显示审核任务列表，包括所属的项目名称、项目版本、发布时间等信息； 3. 审核人员刷新任务列表； 4. 系统显示最新的审核任务列表。
扩展流程	1a. 审核人员选择“查看历史审核任务”； 1. 系统显示审核人员之前已审核的任务信息。

查看审核任务列表用例描述如表 3.3 所示，审核人员在登录后即可查看自己被分配的审核任务，快速了解任务的概要信息。同时系统可能会根据后台主动学习模型的训练进度推送新的审核任务，当然审核人员也可以选择手动刷新的

方式查看更新的信息。审核人员可以根据自己的时间安排选择任务进行审核流程。审核人员也可以查看自己的审核历史记录。

表 3.4: 查看警告信息用例描述

ID	UC2
参与者	审核人员
触发条件	审核人员选择了一项分配给自己的审核任务
前置条件	审核人员登录系统并通过认证
后置条件	无
优先级	高
正常流程	1. 审核人员点击“查看详情”按钮； 2. 系统显示警告的详细信息，包括警告的优先级、等级、类型、代码行，和对应源代码所在的文件路径、类名、方法名、方法签名等信息，并展示代码片段。
扩展流程	无

查看警告信息的用例描述如表 3.4所示，审核人员从审核任务的列表中选择一条记录，查看其具体的信息。如果该警告信息所属的审核任务是在主动学习的查询阶段生成的，还将提供当前系统对于此警告是正报的概率判断，供审核人员参考。

表 3.5: 填写审核结果用例描述

ID	UC3
参与者	审核人员
触发条件	审核人员已经阅读了警告相关信息
前置条件	审核人员登录系统并通过认证
后置条件	系统保存审核人员对警告的审核结果
优先级	高
正常流程	1. 审核人员选择审核结果：正报或者误报； 2. 系统提示是否确认保存结果； 3. 审核人员选择“确认保存”； 4. 系统将结果保存至服务器数据库。
扩展流程	3a. 审核人员选择“取消”； 1. 系统取消保存操作。

填写审核结果的用例描述如表 3.5所示，审核人员将结果提交反馈给系统。

为防止手误提交，还引入了二次确认的流程，并且不提供选项的默认值，使审核人员必须仔细审核后才能选择提交。

表 3.6: 账户注册用例描述

ID	UC4
参与者	审核人员
触发条件	审核人员点击“注册”按钮
前置条件	无
后置条件	系统保存审核人员的账户信息
优先级	高
正常流程	1. 审核人员填写所需信息，包括用户名、邮箱和登录密码； 2. 系统将账户信息保存至服务器数据库。
扩展流程	2a. 系统显示用户名已存在； 1. 审核人员重新填写表单。

账户注册的用例描述如表 3.6所示，新加入的审核人员可以填写用户名、密码、邮箱等信息注册成为系统的用户。在注册过程中如果遇到问题，系统会以文字提示的方式显示在界面上，并提示用户进行下一步操作。

表 3.7: 查看项目列表用例描述

ID	UC5
参与者	管理员
触发条件	管理员成功登录或点击“项目列表”返回首页
前置条件	管理员登录系统并通过认证
后置条件	界面显示项目列表
优先级	高
正常流程	1. 管理员登录成功后自动跳转或点击“项目列表”返回首页； 2. 系统返回所有项目的信息，包括项目的名称、版本、上传时间、标记完成时间、当前状态等信息。
扩展流程	无

查看项目列表的用例描述如表 3.7所示，管理员在登录后将看到当前系统上保存的所有项目的信息。其状态可以分为：扫描中、审核中和审核完毕三种。如果项目处于正在审核中的状态，会显示当前的整体审核进度。列表将按一定

顺序排列，正在审核的项目会排在前面，管理员可以直观地看到当前进行检测任务，从而决定是否前往查看报告页面查看最终的结果。

表 3.8: 上传项目用例描述

ID	UC6
参与者	管理员
触发条件	管理员点击“上传项目”按钮
前置条件	管理员登录系统并通过认证
后置条件	系统将项目相关文件保存至服务器，相关信息保存至数据库
优先级	高
正常流程	1. 管理员填写项目名称、版本和描述，从本地选择源代码压缩包和编译打包产物并提交； 2. 系统检查相关信息并返回提交结果。
扩展流程	2a. 系统显示项目已存在； 1. 管理员重新提交。

上传项目的用例描述如表 3.8所示，管理员需要启动一个新的检测任务时可以使用上传项目的功能，将代码打包后上传至服务器。之后系统会自动进行扫描，并使用主动学习模型进行查询标记审核分发的任务。

表 3.9: 查看最终报告用例描述

ID	UC7
参与者	管理员
触发条件	管理员点击“查看报告”按钮
前置条件	管理员登录系统并通过认证
后置条件	系统显示指定的项目检测报告
优先级	高
正常流程	1. 管理员点击显示项目名称和版本列表的下拉选择框； 2. 系统返回已审核完成的所有项目列表； 3. 管理员选择项目并点击“查看”按钮； 4. 系统返回该项目下所有警告的详细结果。
扩展流程	无

查看最终报告的用例描述如表 3.9所示，系统管理员在看到项目的检测以及审核工作均已经完成时，可以查看最终的结果。在选择对应的项目版本后，系

系统将给出静态扫描工具从项目中检测出的警告列表和审核人员的标记结果。

表 3.10: 审核人员管理用例描述

ID	UC8
参与者	管理员
触发条件	管理员点击“人员管理”按钮
前置条件	管理员登录系统并通过认证
后置条件	系统完成指定操作
优先级	高
正常流程	1. 管理员点击“人员管理”按钮； 2. 系统返回当前系统所有审核人员的信息，包括名字、邮箱和已审核漏洞。
扩展流程	1a. 管理员选择停用审核人员的账户； 1. 系统返回停用操作的处理结果。

审核人员管理的用例描述如表 3.10 所示，在这个用例中，管理员可以看到当前系统中已注册的审核人员以及他们的审核完成情况，以便及时督促他们完成工作。此外，当团队成员发生变化时，管理员可以停用审核人员的账号，以保证系统的安全性。

3.3 系统架构设计

从整体上来看，系统的架构设计如图 3.3 所示，采用了前后端分离的架构设计，前端应用和后端服务器之间以 JSON 格式进行数据传输和交互。这摆脱了传统的 Java Web 开发中，前端代码和后端代码混杂在 JSP 页面的糟糕情形。前后端分离使得两者可以分开开发，不必紧密地耦合在一起。其中，前端独立地编写网页页面代码，通过 HTTP 请求来访问后端定义的数据接口；而后端独立编写服务端代码提供数据接口，甚至可以使用不同的语言开发不同模块的接口。在测试时只需按照接口文档的规定，构造测试用的 HTTP 假数据请求即可。前后端分离的开发方式也有利于后期的系统集成。

在本文的系统中，前端使用 Vuejs 框架开发页面，后端服务使用 Python 语言下的 Flask 框架开发，在以 Restful API 提供服务接口的同时也能方便地调用主动学习模型。在特征提取和漏洞扫描的过程中，需要在 Python 中以命令行方式调用 Java 的 jar 包，通过读取其在命令行的输出获得结果。在实际开发过程

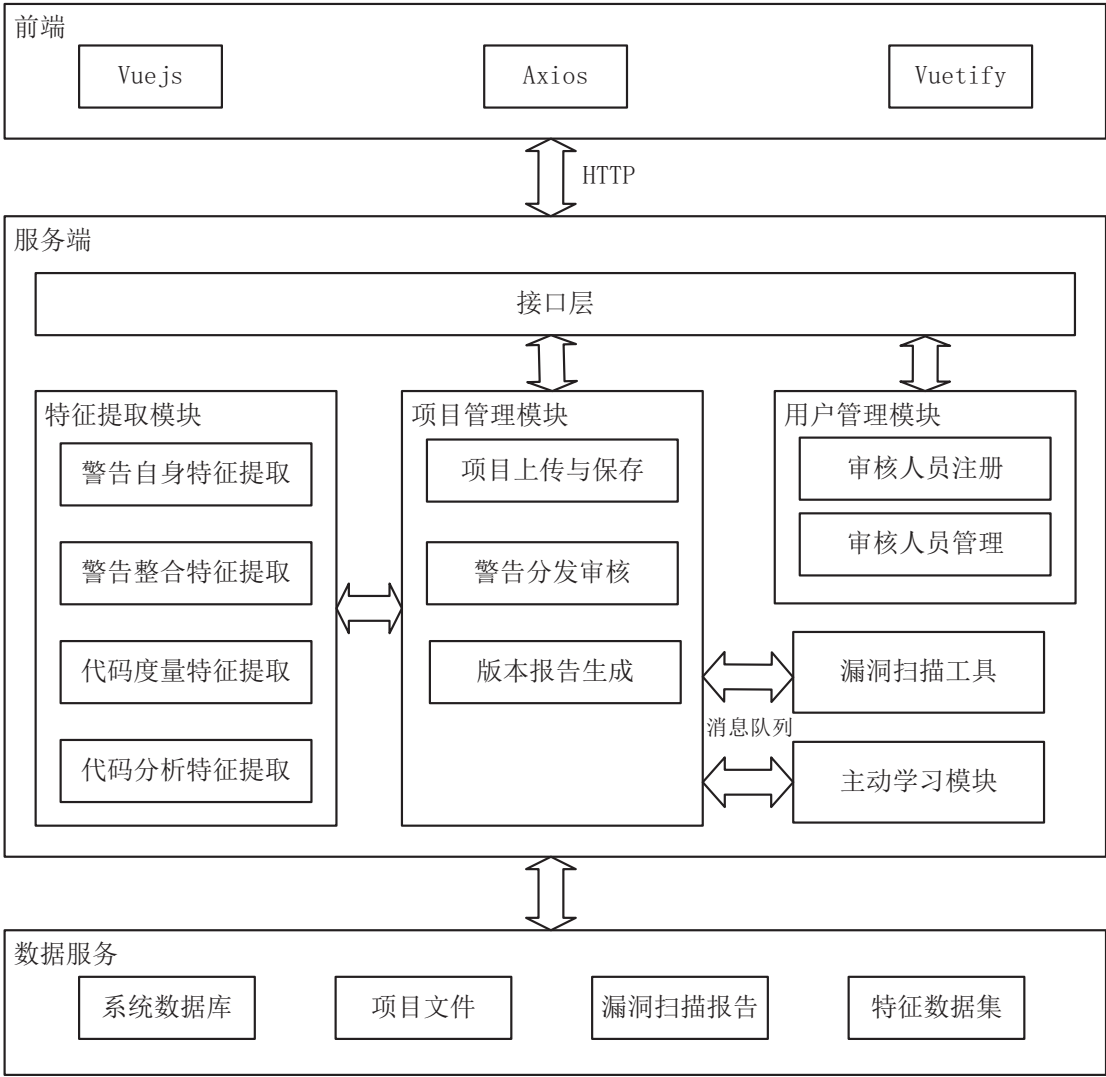


图 3.3: 系统架构图

中，还采用了 Git 实现代码的版本控制。

1. 使用 `Vuejs` 框架进行前端开发，利用其组件化开发的思想将标题工具栏设计为可复用的组件，利用其数据驱动双向绑定的思想将组件状态和显示内容全部与数据相绑定，数据的变化即可引起界面元素的变化。界面的 UI 风格和组件由 `Vuetify` 提供，可以快速构建简单清晰符合 `Material`^①设计的界面。组件的行为采用 `JavaScript` 编写，并使用了 `Axios` 构造 `HTTP` 请求与后端通信。前端的打包工作由 `npm`^②负责，打包为单页面应用并放置在服务端的资源文件夹下。

^①<https://material.io>

^②<https://www.npmjs.com>

2. 服务端采用 Python 语言的 Flask 框架，按照传统的 Controller-Service-Dao 三层模型进行开发。其中 Controller 层负责对请求的解析和校验，利用 session 进行权限控制，以及调用相应的 Service 获得结果并包装成 HTTP 响应返回。Service 层封装了多种业务逻辑，根据传入的参数调用对数据库不同表的操作，通过逻辑判断或数据组合返回结果。Dao 层负责数据库操作，使用 peewee 作为数据库 ORM 框架，可以方便地实现数据的新增、更新、删除和查询操作。
3. 主动学习模块采用 sklearn 包实现机器学习，numpy 和 pandas 包实现数据读取与处理，在循环中不断改进机器学习模型的训练效果，并向服务端程序提供待审核警告以及预测概率。由于主动学习模块的执行时间较长且需等待审核人员返回标记结果，因此相关操作被包装为单独的线程处理。
4. 静态代码分析工具与主动学习模块都不会立即返回执行结果，因此需要异步执行。为了防止系统的主要服务被阻塞，系统采用 RabbitMQ 消息队列作为他们模块之间的通信桥梁。虽然需要安装配置额外的 RabbitMQ 服务器，但是其实现的发布/订阅模式实现了服务之间的解耦，在执行异步操作时提升了服务器的响应速度和性能。
5. 考虑到本系统规模并不庞大，因此使用 SQLite 这一轻量级数据库。SQLite 不需要过多的配置即可使用，它将数据库存储在单一的磁盘文件上，而不需要运行一个单独的数据库服务器进程。

3.4 系统视图模型

软件作为一种复杂系统，其架构设计无法仅从一个方面概括，因此需要使用 4+1 视图作为自顶向下的设计分解方法。其中的“4”指逻辑视图、开发视图、进程视图和物理视图。“1”是指用例视图，也称为场景视图，在上文需求分析的用例模型描述一节已介绍过。其余四种视图的作用如下：

1. 逻辑视图：使用面向对象的思想对系统职责逐级划分，并描述各逻辑元素间的关系。系统的各个模块划分应该满足“高内聚，低耦合”的原则，从逻辑视图中可以看出系统是否存在着不合理的模块划分和依赖关系。
2. 开发视图：开发视图用于描述系统在开发时的逻辑结构，侧重于对软件模块的组织和管理。
3. 进程视图：通过时序图描述系统运行时与用户的交互行为，以及各组件之

间的交互关系。在大型系统中可能还涉及并发与抢占。

4. 物理视图：反映了系统各逻辑组件在实际物理设备的分布情况。

3.4.1 逻辑视图

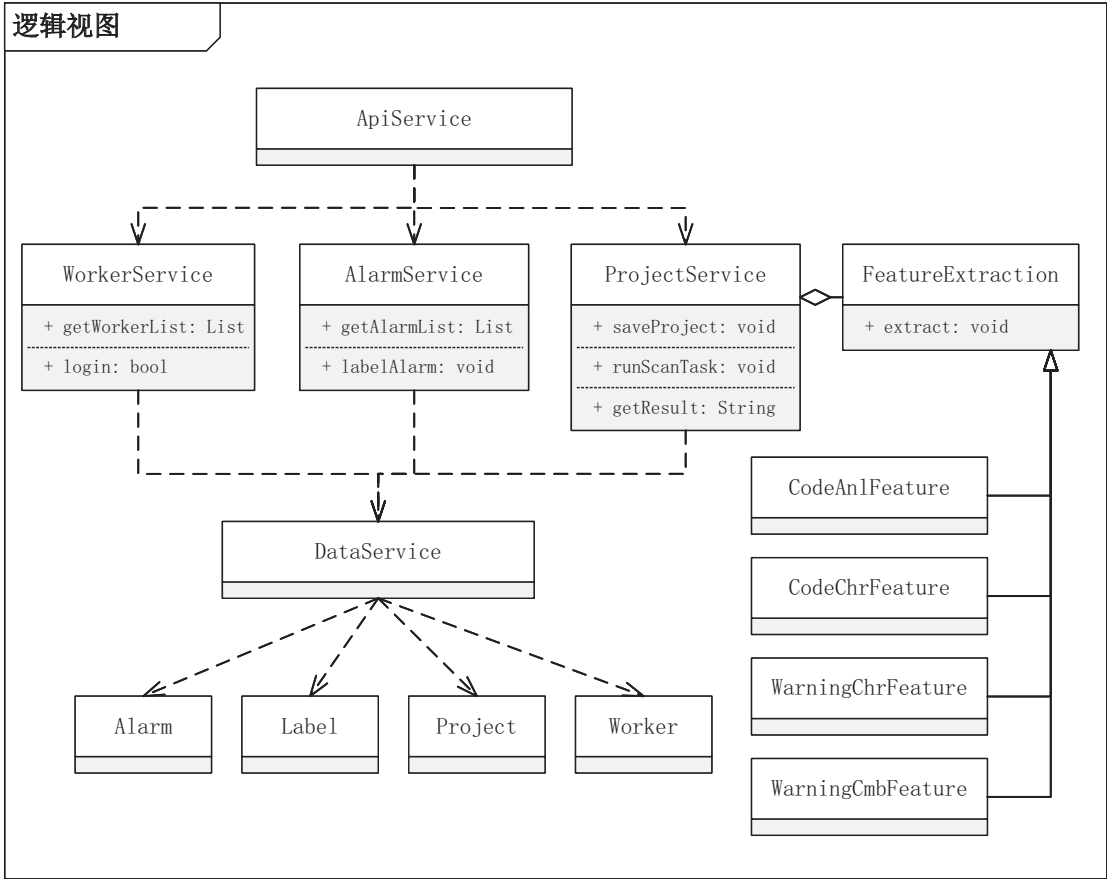


图 3.4: 系统逻辑视图

依据本文系统架构设计中的功能模块划分，绘制了如图 3.4 所示的系统逻辑视图。ApiService 负责接收并处理前端的请求，并根据需要调用其关联的 WorkerService、AlarmService 或者 ProjectService。被调用的这三个 Service 都与负责数据库服务的 DataService 相关联。DataService 调用四个数据库模型类 Alarm、Label、Project、Worker 完成对数据库的操作。其中 ProjectService 在提取特征时会调用一组 FeatureExtraction 类，其具体的方法实现则根据特征的不同角度由 CodeAnlFeature、CodeChrFeature、WarningChrFeature 和 WarningCmbFeature 分别完成。

3.4.2 开发视图

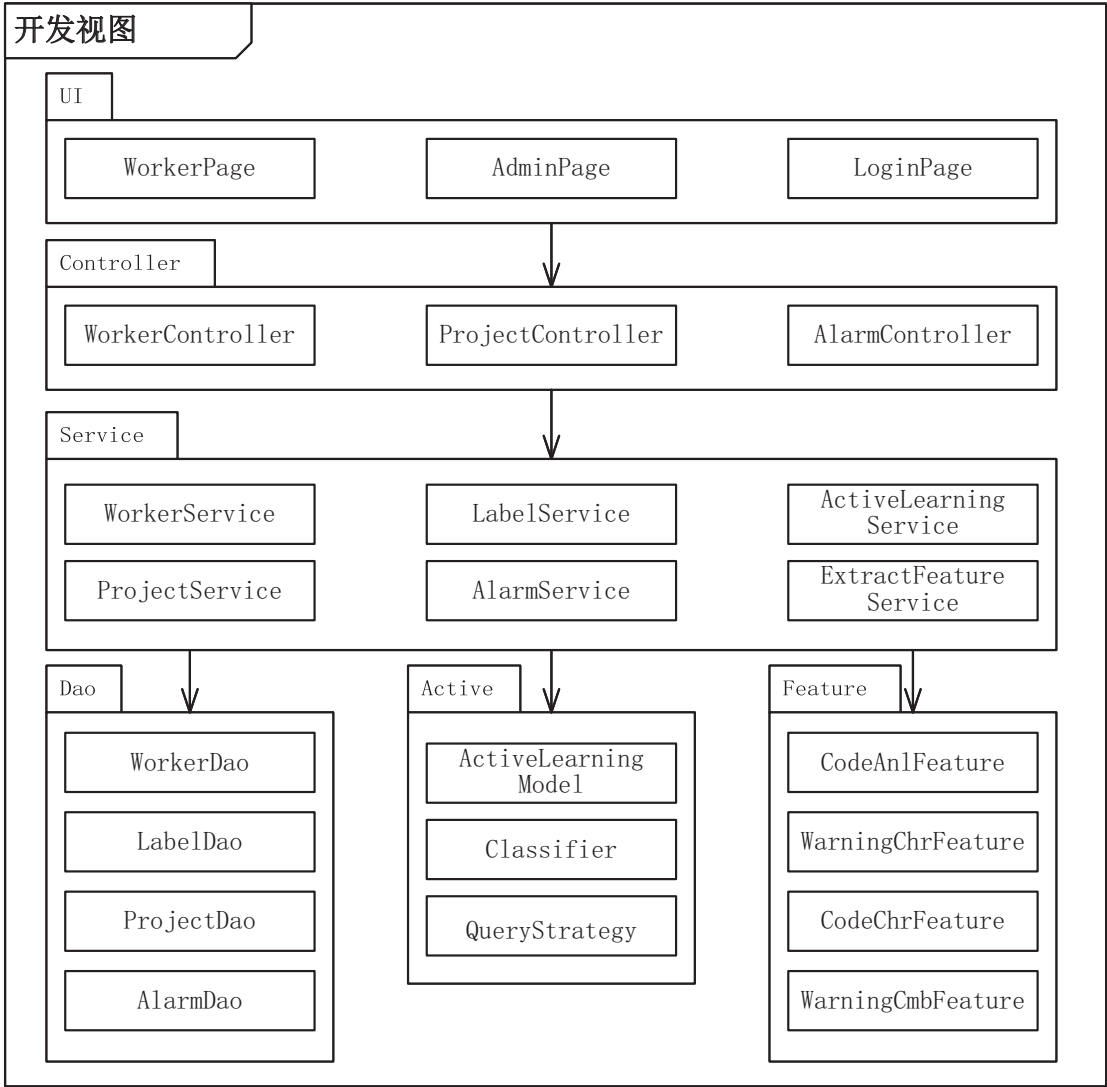


图 3.5: 系统开发视图

图 3.5描述了系统的开发视图。开发视图的 UI 包位于前端，除了登录页面 LoginPage 之外，依据使用者角色的不同划分为审核人员界面 WorkerPage 和管理员界面 AdminPage。Controller 包、Service 包和 Dao 包分别对应着服务端架构设计中的三层模型。此外，Active 包中包括主动学习模型、集成学习实现的分类器以及不同的查询策略。Feature 包中包含了几种不同角度的特征提取工具类，与上文逻辑视图中相对应。

3.4.3 进程视图

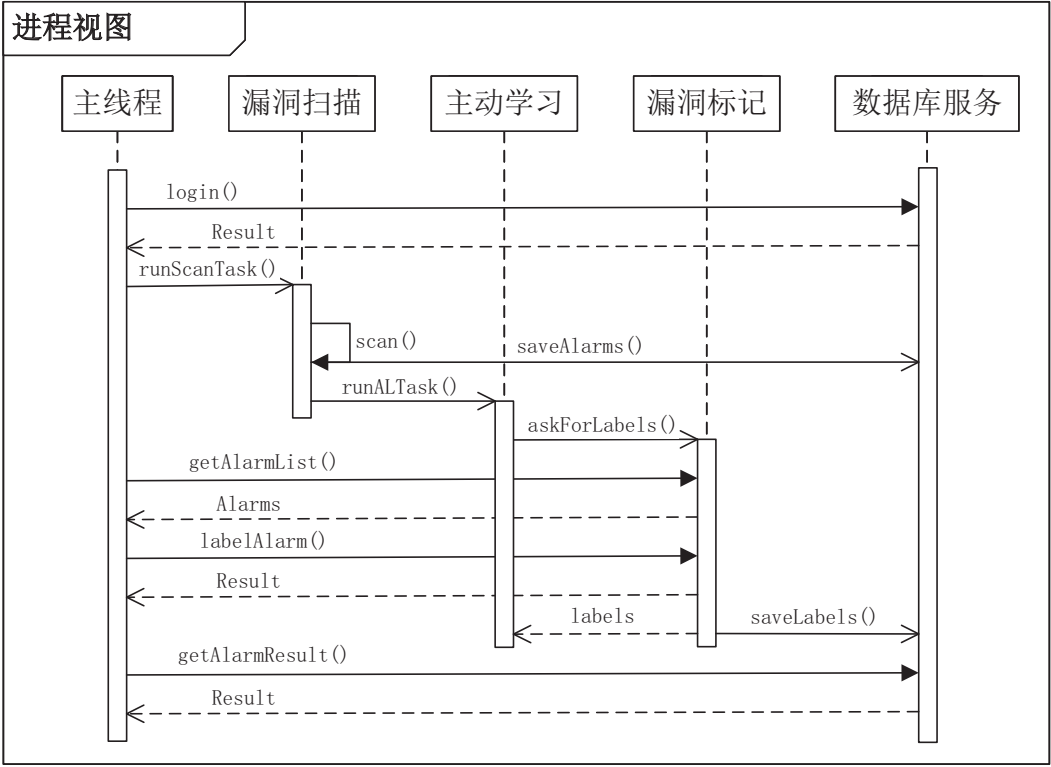


图 3.6: 系统进程视图

图3.6描述了系统的进程视图。首先是在用户登录时访问数据库并获取认证的结果。当主线程收到管理员上传的项目时会调用 `runScanTask` 以异步的方式开始漏洞扫描，扫描结束后将扫描报告存入数据库，并立即调用 `runALTask` 启动主动学习的线程。主动学习的线程运行过程中，会向漏洞标记线程请求警告的标签。这时主线程会通过 `getAlarmList` 方法获取所有待标记的漏洞，拿到返回的结果后再通过 `labelAlarm` 方法提交审核人员的标记结果。当标记线程完成所有标记任务时，会向主动学习线程返回标记的结果并将其持久化到数据库。最后，主线程可以通过 `getAlarmResult` 方法获取到所有标记完成的警告列表。

3.4.4 物理视图

系统部署视图如图3.7所示。用户在客户端上使用 Web 浏览器与系统交互，通过 HTTP 请求与部署在防火墙后面的服务器通信。在后端服务器上存在多个相对独立的应用程序，分别是负责处理 Web 请求的主程序，主动学习模型

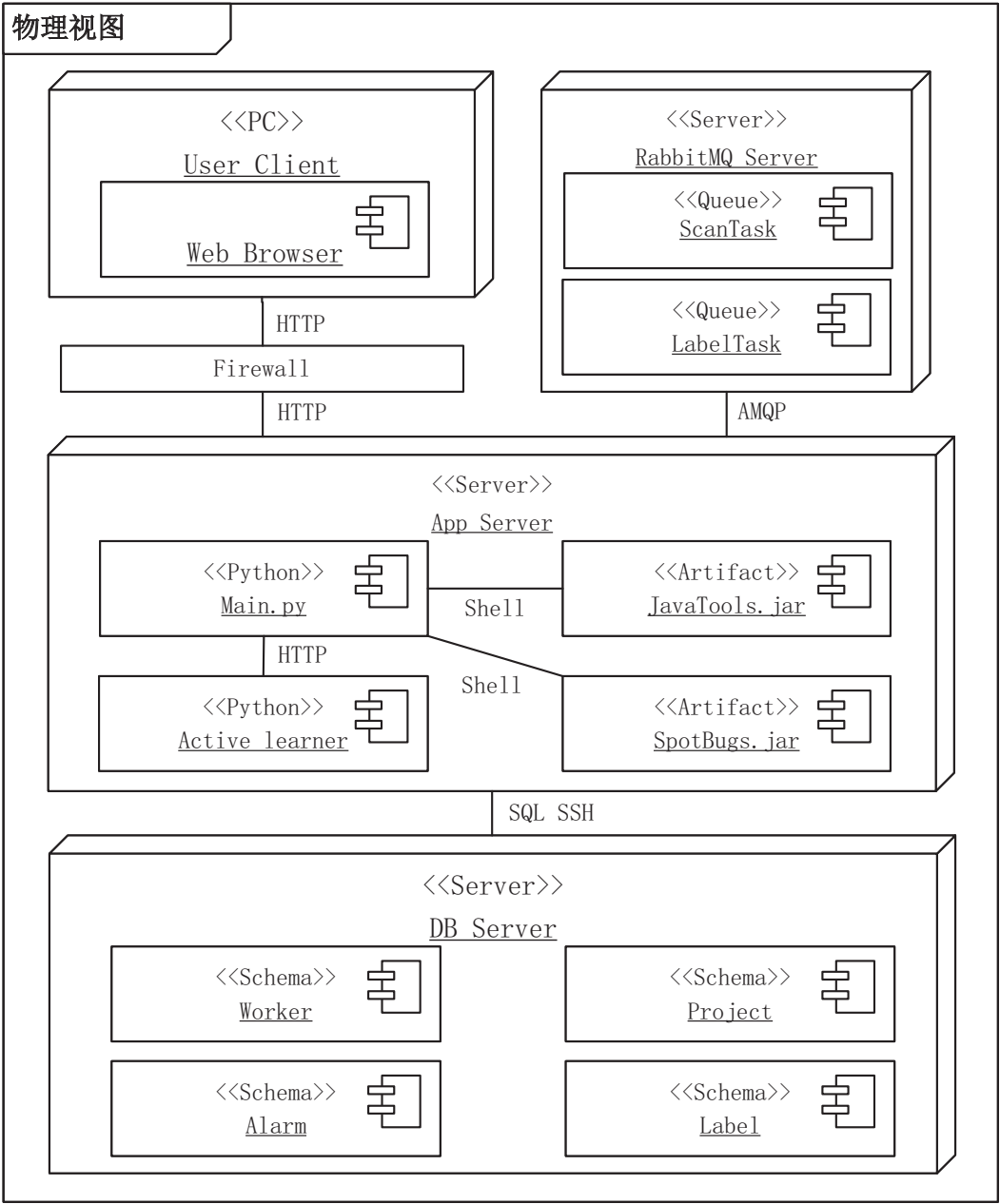


图 3.7: 系统物理视图

线程，基于静态分析的漏洞扫描工具和处理源代码 AST 的工具程序。主程序会以执行 Shell 命令的方式调用 Jar 包并获取执行结果。而主程序与其他线程之间会通过消息队列的 AMQP 协议进行通信，需要发布和订阅的是 ScanTask 和 LabelTask 两个队列。数据库服务器用于持久化存储系统使用的数据，包括 Worker、Project、Alarm 和 Label 四张表。

3.5 数据库设计

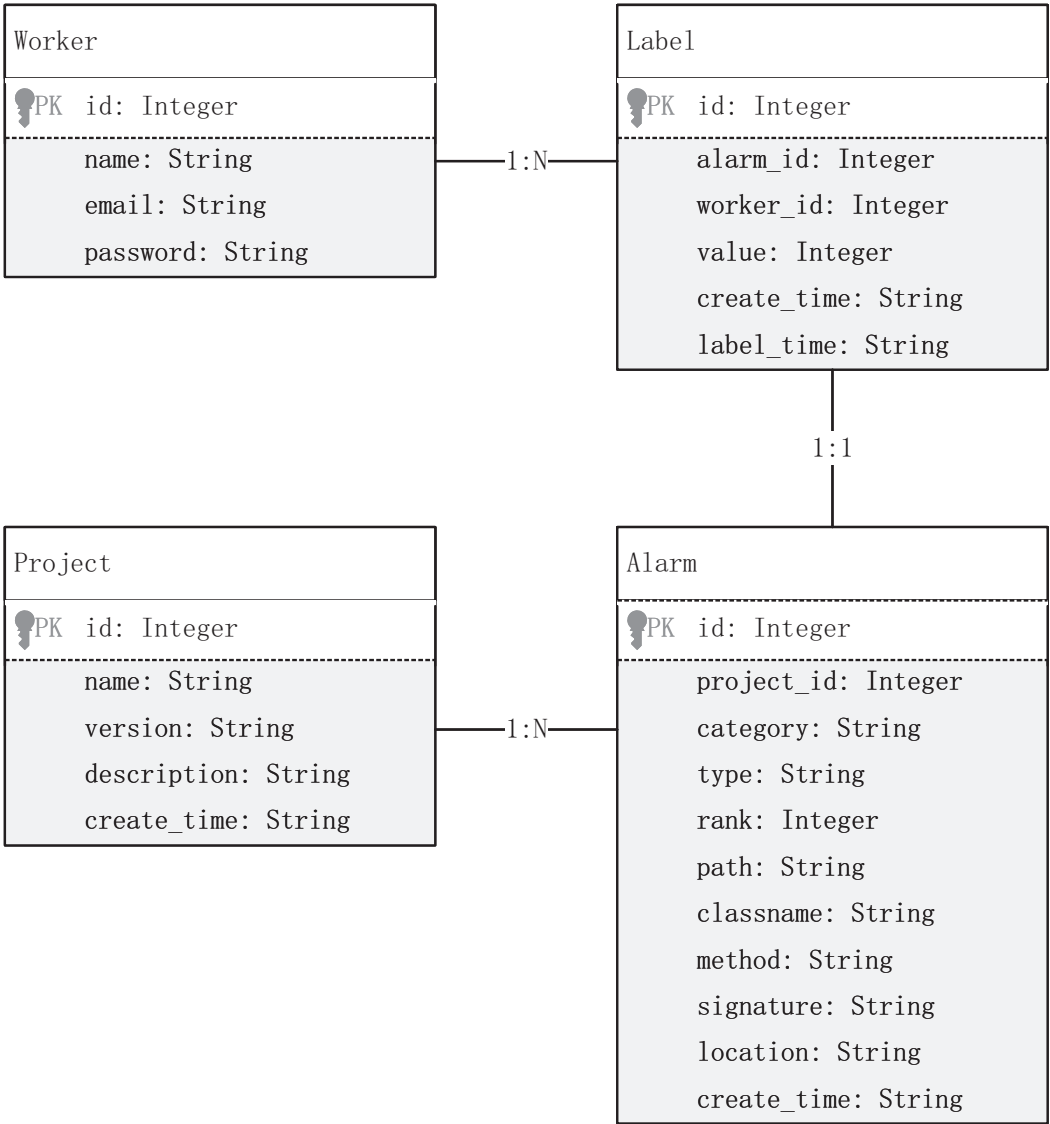


图 3.8: 系统实体类设计图

在本文设计的系统中，需要持久化的实体类包括审核人员类 **Worker**、项目类 **Project**、警告类 **Alarm** 和警告标记类 **Label**。实体类和之间关系的设计图如图 3.8 所示。一个项目可以包含 0 到 N 个警告，由于警告只会分配给一位审核人员标注，因此一个警告对应一个警告标记类，但是审核人员可以标记多个不同的警告，因此审核人员对应着多个警告标记类。下面的表格中列出了每个数据表实体类的字段、类型、属性及其表示的含义。属性一列中，PK 表示主键，AI 表示数据值自增，NN 表示字段非空，NQ 表示字值不能重复。

表 3.11: Worker 类设计描述

字段	类型	属性	含义
id	Integer	PK, AI	主键
name	String	NN, NQ	审核人员姓名
email	String		注册邮箱
password	String	NN	登录密码

Wroker 类设计描述如表 3.11 所示。审核人员的姓名 name 字段被设置为非空且不可重复，登录密码 password 字段也为非空，用于登录验证。在注册新用户时，如果输入的姓名与已有的重复，则禁止其注册。注册邮箱 email 字段可以选择不填写。

表 3.12: Alarm 类设计描述

字段	类型	属性	含义
id	Integer	PK, AI	主键
project_id	Integer	NN	所属项目 ID
category	String	NN	警告分类
type	String	NN	警告类型
rank	Integer	NN	优先级
path	String	NN	文件路径
classname	String	NN	类名
method	String	NN	方法名
signature	String	NN	方法签名
location	Integer	NN	代码行位置
create_time	String	NN	扫描时间

Alarm 类的设计描述如表 3.12 所示，project_id 字段指向 Project 表中的一条记录，实现项目与警告之间的关联。本表中的大部分字段均来源于静态分析工具的扫描报告 xml 文件，为了便于查询，需要将它们持久化存储在数据库中。表中的代码行指经过格式化处理后对应的代码行，保存时需要替换处理。

Project 类的设计描述如表 3.13 所示，其中项目的名称和版本号共同确定了一个唯一的项目。项目的状态采用整数表示，0 表示正在检测中，1 表示正在审核中，2 表示已经审核完成。状态值的存在使得不必每次查询时都计算得到当前的状态，加快了查询和响应的速度。上传时间字段将在新增记录时以当前日

表 3.13: Project 类设计描述

字段	类型	属性	含义
id	Integer	PK, AI	主键
name	String	NN	项目名称
version	String	NN	项目版本
description	String		项目描述
state	Integer	NN	状态
create_time	String	NN	上传时间

期时间填写。

表 3.14: Label 类设计描述

字段	类型	属性	含义
id	Integer	PK, AI	主键
alarm_id	Integer	NN	警告 ID
worker_id	Integer	NN	标记者 ID
value	Integer	NN	警告标记
create_time	String	NN	任务创建时间
label_time	String	NN	标记时间

Label 类的设计描述如表 3.14 所示，记录了警告的 ID 和审核人员的 ID。虽然警告和警告标记是一一对应的关系，但是依然选择设计成分表存储的方法，这是因为 Alarm 内的数据创建后几乎不会修改，而警告的标记会被修改。同时如果之后需要将一个警告分配给多个审核人员标注，这样设计可以很好地应对可能出现的变化。

3.6 系统详细设计

系统的详细设计阶段需要为之后的实现提供更多细节，例如确定关键的核心类、确定接口的细节等。本节将描述对系统的详细设计，总体上采用结构化程序设计方法，如有必要将使用适当工具对涉及的算法进行描述。最终实现改善控制结构，降低程序复杂度，获得正确可靠的模块描述。

为了便于后期的开发实现，本文设计的系统可划分为几个主要模块，以下概括了各模块对应的功能点：

- 项目管理模块：查看项目列表、上传保存项目、对项目进行漏洞扫描；
- 特征提取模块：多个角度的特征提取与保存；
- 主动学习模块：主动学习模型的训练与使用；
- 警告审核模块：查看警告列表、审核任务分发、保存审核结果。

3.6.1 项目管理模块

项目管理模块主要负责实现管理员对系统内项目的管理操作，以及在项目上传时保存项目相关的数据。目前系统仅能处理以 Java 语言编写的软件开发项目，因此在项目上传时需要加以限制。项目上传成功是系统操作流程的第一步，后续才会开始漏洞扫描和特征提取等一系列流程。

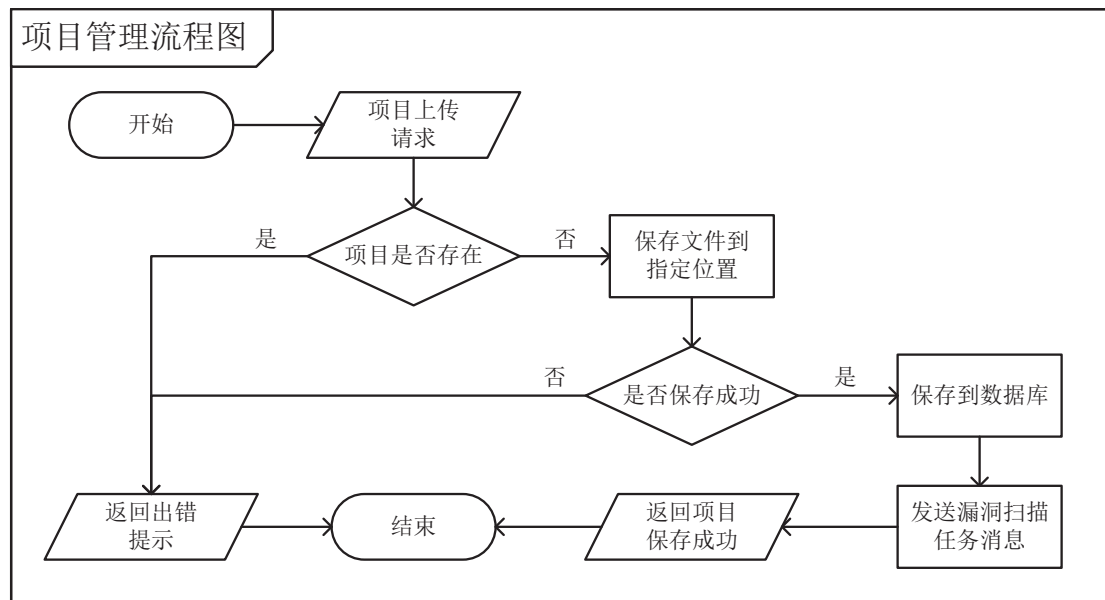


图 3.9: 项目管理模块流程图

本模块的主要执行流程如图 3.9 所示。用户提交上传请求之后，系统解析请求中包含的表单，根据填写的项目名称和版本号判断项目是否已经存在于系统中。如果此项目不存在，则创建新的目标文件夹后将文件保存到指定位置。保存过程中可能会出现问题，比如磁盘空间不足，因此保存后需要测试是否保存成功，如果保存成功再将项目数据持久化到数据库。接着向消息队列的特定 Channel 中发送消息，包含了扫描位置和扫描报告保存位置等参数。最后根据上述流程的执行结果构造响应，返回给用户。

本模块的核心类图设计如图 3.10 所示，其中 ProjectService 是向接口层提供

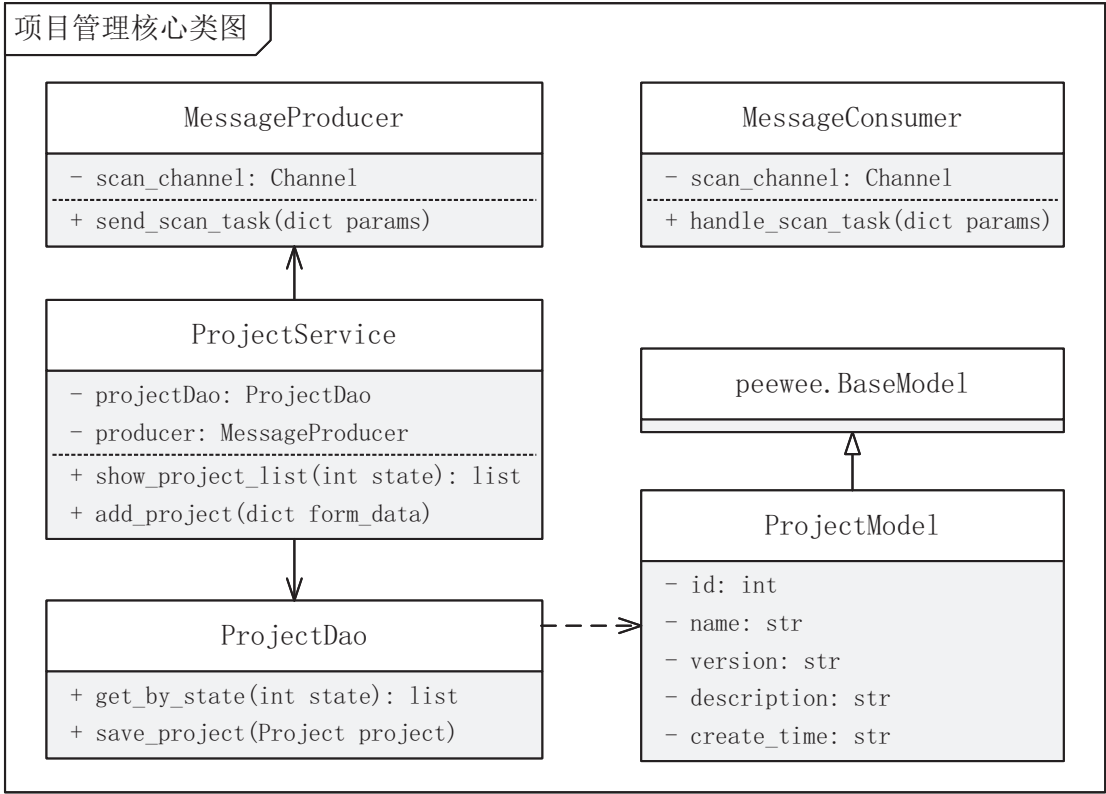


图 3.10: 项目管理模块核心类图

服务的服务类，拥有一个 **ProjectDao** 实例和一个 **MessageProducer** 实例。当需要进行数据库操作时，会调用 **ProjectDao** 中的方法。**ProjectDao** 利用了继承了 **peewee** 包提供的 **BaseModel** 基类的 **Project** 类，将对数据库的操作映射为对实体类 **Project** 的操作，通过 **get_by_state** 方法根据项目状态筛选状态（也可以不填写状态值获取全部项目），以及通过 **save_project** 方法保存项目信息。

MessageProducer 类会在初始化时声明一个 **Channel**，并在 **Channel** 中注册一个名为 **scan_task** 的队列。当 **send_scan_task** 方法被调用时，会将参数以 **json** 字符串的形式发布在这个队列中。另一边的 **MessageConsumer** 会以同样的方法声明 **Channel** 并注册队列，当队列收到消息时，会取出消息并解析参数，通过系统命令执行启动静态代码分析工具的 **Shell** 命令，扫描结束后会将报告保存在指定的路径。

本模块的时序图如图3.11所示，首先用户需要向系统中上传项目，在前端页面中填写必需的信息后点击上传按钮，**ProjectController** 将接受请求。之后调用 **ProjectService** 中的方法，持久化数据并向消息队列发送消息。注意消息的生产者 **MessageProducer** 以及消息队列服务器会在系统启动时就创建，在需要时

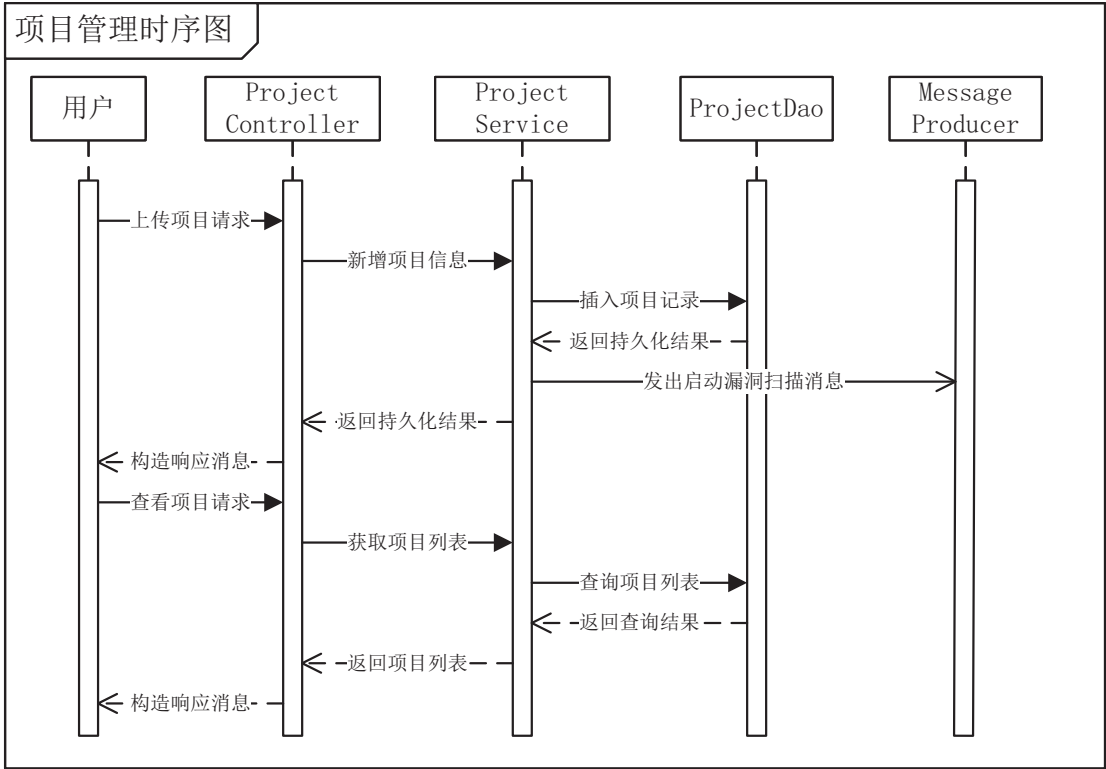


图 3.11: 项目管理模块时序图

其中发送消息的方法会被调用。最后根据操作执行的结果会构造相应的响应数据返回给用户。系统中有项目存在时可以发出查看项目列表的请求，与之前的顺序类似，逐层调用并处理返回的结果。

3.6.2 特征提取模块

特征提取模块负责从漏洞扫描工具的检测结果和源代码文件中挖掘特征，并将特征保存在 csv 文件中以备模型使用。系统的特征并不是固定的，在之后的改进中可能引入更多的特征，因此设计时需要注意满足一定的可扩展性。在使用特征提取模块之前，系统应当已经获得了项目的源代码，和漏洞扫描工具的扫描报告，本模块的输出为一个包含所有警告特征数据的 csv 文件。

本模块的流程设计如图 3.12 所示。当特征提取模块被调用时，首先检查前置条件是否具备，包括项目的源代码文件是否存在，以及扫描报告 xml 文件是否存在。如果前置条件不满足则抛出异常，交由调用者处理。提取特征时，特征被划分为不同的种类，不同大类下的特征提取都是相互独立的，因此可以以任意顺序执行不同的特征提取方法。这里涉及的特征提取方法包括：提取警告

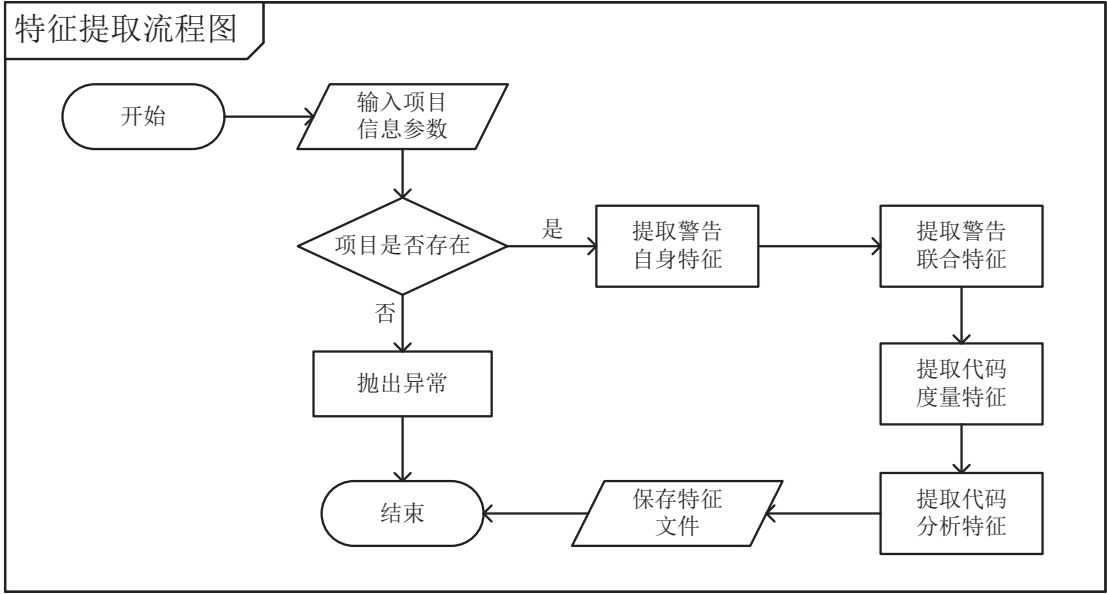


图 3.12: 特征提取模块流程图

自身特征、提取警告联合特征、提取代码度量特征和提取代码分析特征。最后将这些特征结果合并成一个 **DataFrame**，并以 **csv** 文件格式保存至指定路径。

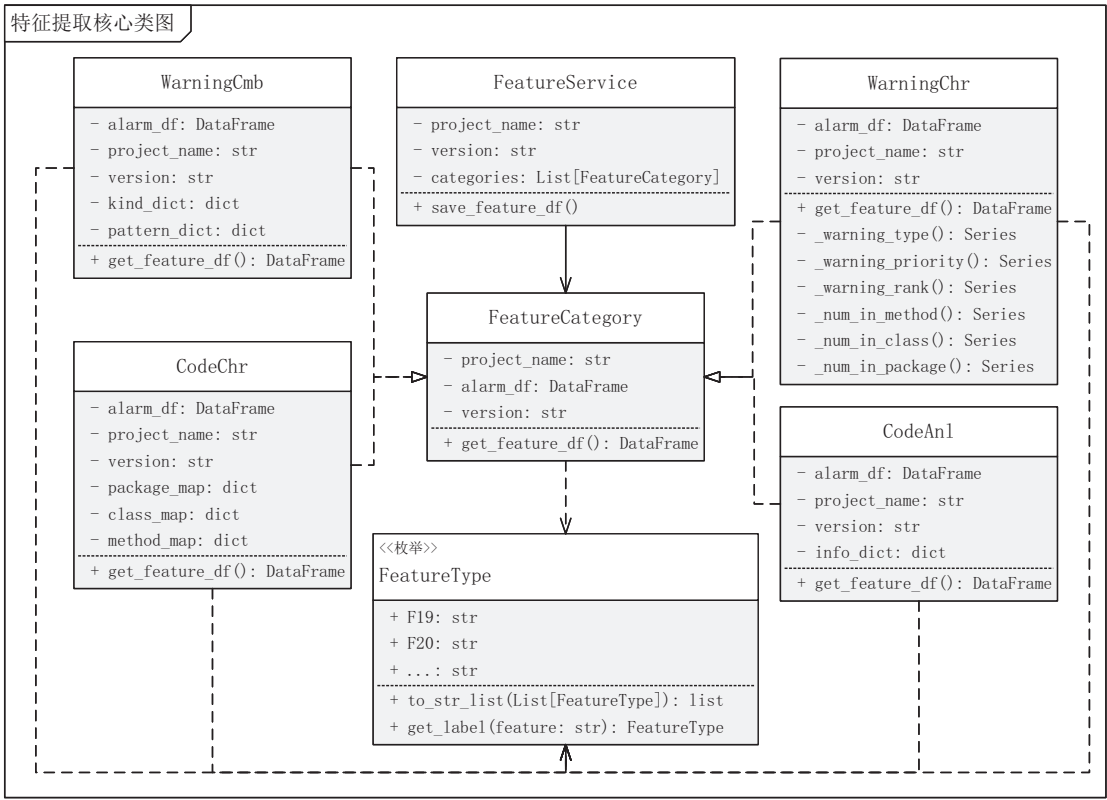


图 3.13: 特征提取模块核心类图

本模块的核心类设计如图 3.13 所示。考虑到之后需要方便地增加新的特征种类，抽象出了一个 `FeatureCategory` 抽象类作为基类，存有公共的属性以及需要实现的 `get_feature_df` 接口。其中 `project_name` 和 `version` 字符串用于确定项目的路径，`get_feature_df` 会返回一个以警告 ID 为索引，每一列均为一种特征值的 `DataFrame`。在 `FeatureService` 类中有一系列 `FeatureCategory` 的引用，其具体的方法分别由 `WarningCmb`、`WarningChr`、`CodeChr` 和 `CodeAnl` 实现。当保存特征的方法被调用时，会依次调用它们的 `get_feature_df` 方法，并将返回结果拼接到一起。此外设计了一个枚举类用于保存特征类型的简写和对应的字符串值，并提供了相应的转换方法。

在每个 `FeatureCategory` 的实现子类中可以有子类独有的成员变量，用于存放特征提取过程中的中间数据。例如 `WarningCmb` 类中 `kind_dict` 和 `pattern_dict` 分别表示同一漏洞类别和同一漏洞模式下正报的数目，`CodeChr` 中的 `package_map`、`class_map`、`method_map` 分别存放包、类、方法级别下正误报的分布情况。这些中间数据会在 `get_feature_df` 方法中被使用到。

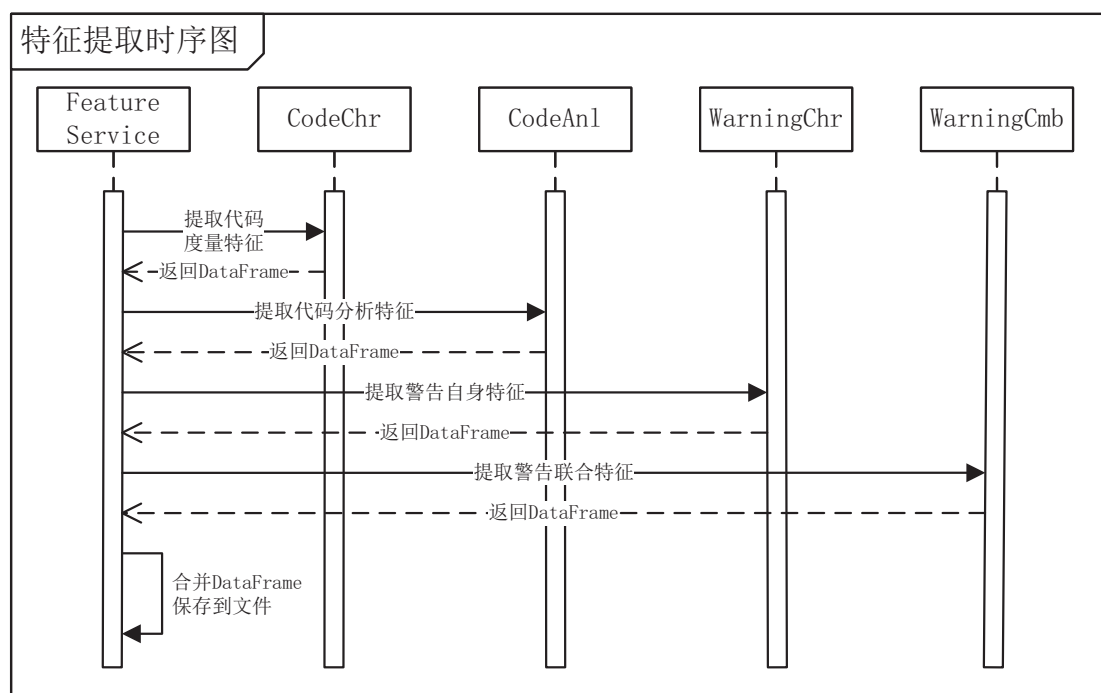


图 3.14: 特征提取模块时序图

本模块的时序图如图 3.14 所示，由 `FeatureService` 方法依次调用 `CodeChr`、`CodeAnl`、`WarningChr` 和 `WarningCmb` 并获取返回 `DataFrame`，最后合并并保存。这里的执行顺序比较简单且上文已经介绍过，故不再赘述。

3.6.3 主动学习模块

主动学习模块用于执行主动学习流程，在循环迭代中训练机器学习模型，配合用户审核模块查询警告的标签。在实际使用中，会将主动学习模型包装在一个单独的线程中运行。主动学习模块执行完毕后，项目的所有警告应当都处于已标记的状态，标记可以是来自人工审核，也可以是来自训练完毕的模型预测的结果。

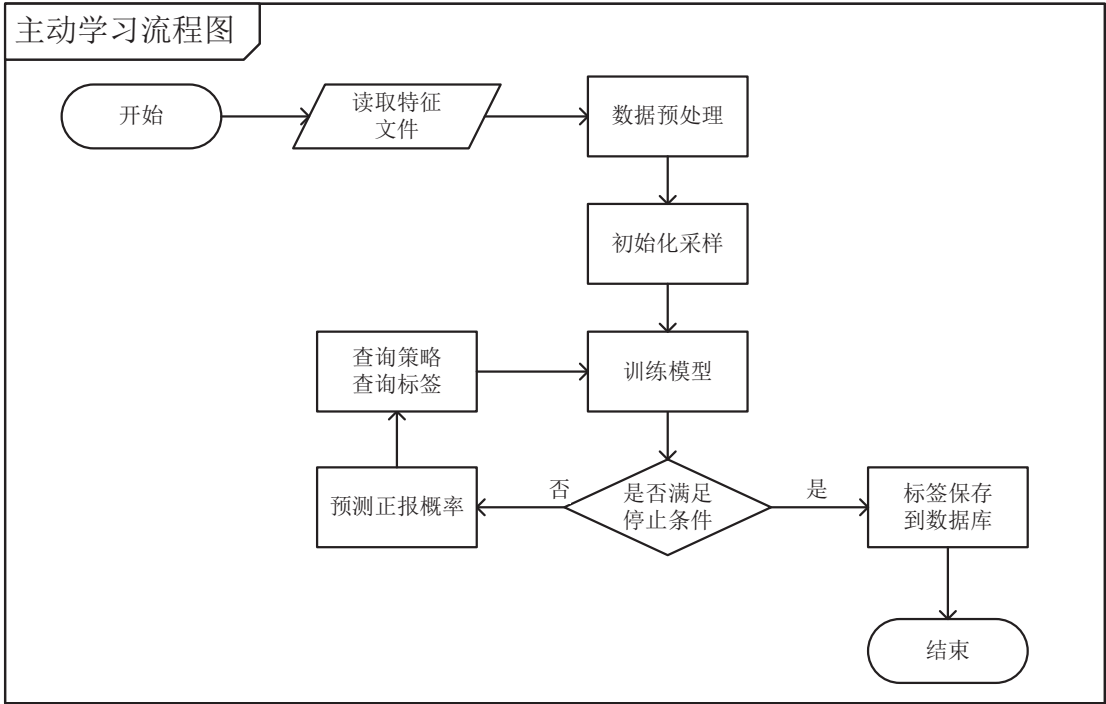


图 3.15: 主动学习模块流程图

本模块的流程设计如图3.15所示。主动学习模块作为一个新的线程启动后，首先准备模型所需的数据，即根据相关配置进行数据的清洗和预处理。然后进行的是初始化采样环节，默认的初始化采样方法是随机抽样，直到抽出的样本中存在着至少一个正报。这是因为如果只有一种标签的数据时，机器学习模型无法进行训练。接下来用已标记的数据训练模型，并使用模型预测出每个实例的正报概率。查询策略会根据这些概率选择合适的样本交给人工审核。最后判断是否满足停止条件，如果满足则退出，否则继续训练机器学习模型。

主动学习模块的核心类设计如图3.16所示，最关键的类是 `ActiveLearning-Model`，包含主动学习模块所需的全部操作。其中持有的 `DataHandler` 类存有预处理配置，提供预处理方法 `preprocess`。 `LabelDao` 负责获取审核标记结果以及

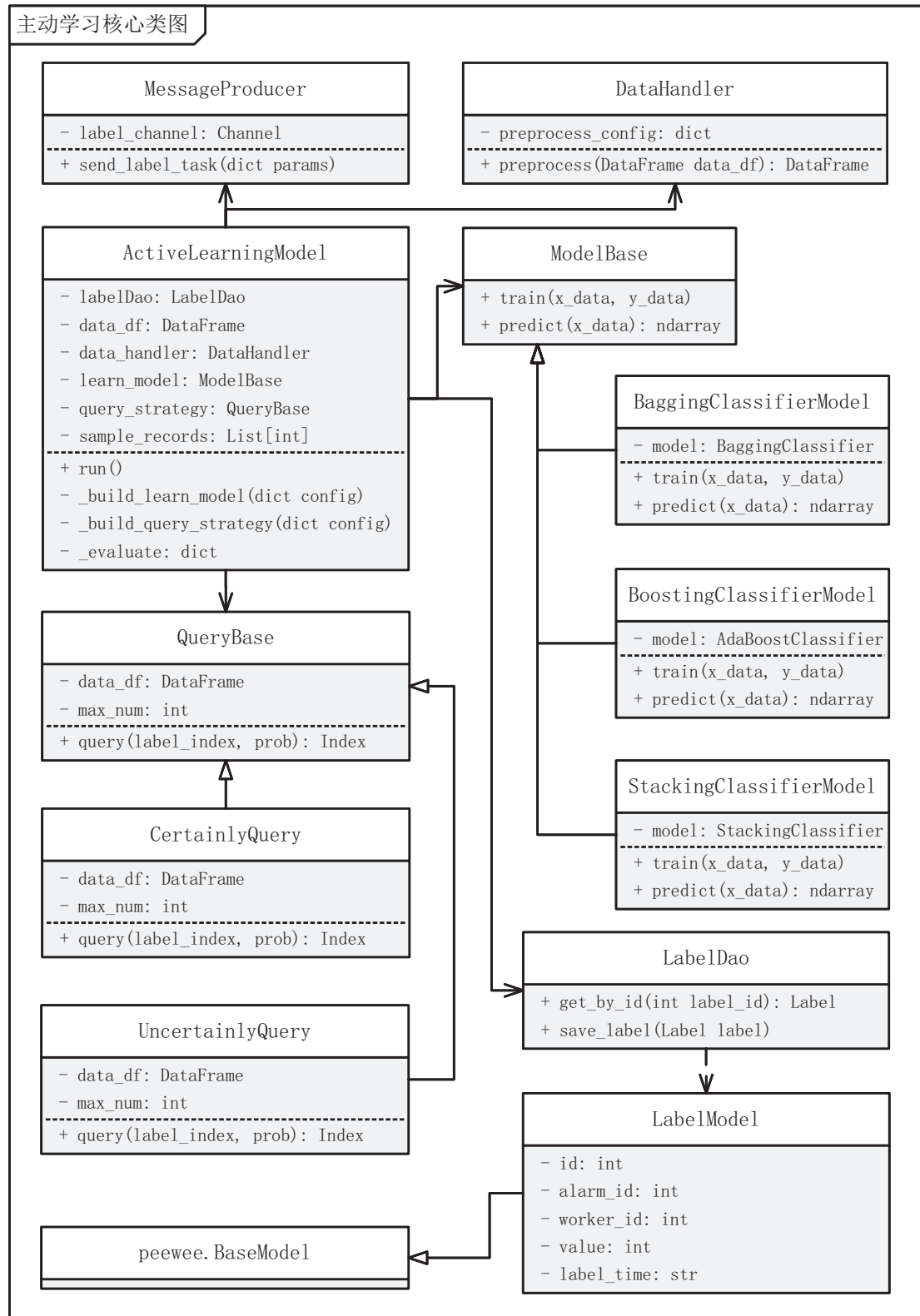


图 3.16: 主动学习模块核心类图

修改审核标记状态。MessageProducer 负责向 label_channel 对应的消息队列中发送审核任务的信息，消息队列的消费者将在警告审核模块处理队列中的消息。LabelModel 映射了对数据库 label 表的操作。

主动学习模块的关键设计在于提供了 ModelBase 和 QueryBase 两个抽象类，分别作为机器学习模型和查询策略的基类。ModelBase 提供了两个待实现方法，train 负责训练模型，而 predict 负责输出模型预测样本为正报的概率，而具体使用哪种模型则由继承抽象类的子类实现。目前对此提供三个不同的实现，分别为 BaggingClassifierModel、BoostingClassifierModel 和 StackingClassifierModel，对应了集成学习中主流的三种方法。QueryBase 保存了整个特征数据表 data_df 和每轮训练结束后的最大查询样本数量 max_num，提供一个查询方法，输入已有审核标记的警告 ID 序列和预测的概率序列，返回下一步需要审核的警告 ID 序列。对此提供了两种不同的查询策略，分别是确定性采样 CertainlyQuery 和不确定性采样 UncertainlyQuery。在主动学习模型初始化时，将调用 _build_learn_model 和 _build_query_strategy 初始化具体的实现类。

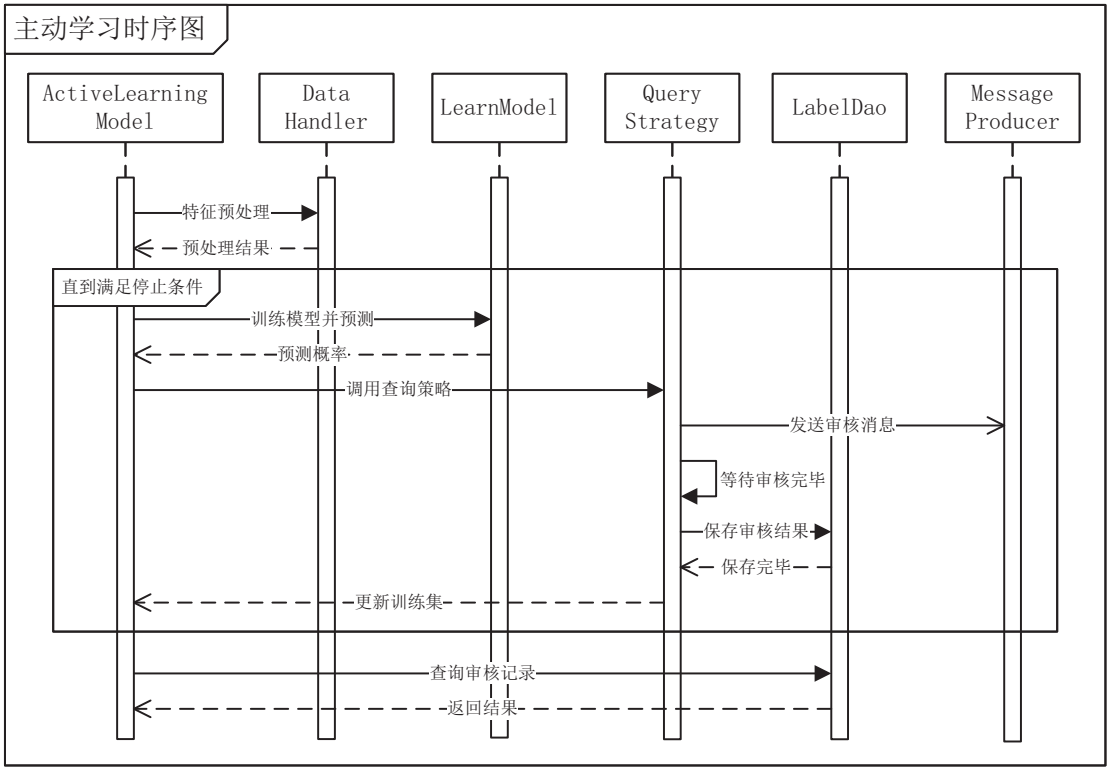


图 3.17: 主动学习模块时序图

主动学习模块的时序图如图3.17所示，在模型创建之初就一次性读入所有

数据样本，统一交由 `DataHandler` 进行预处理并得到其结果，这样不必每次训练前都需要做一次预处理。之后是一个循环片段，将循环进行到满足停止条件为止。首先调用 `LearnModel` 训练机器学习模型并对剩余的未审核警告进行正报概率的预测。接着使用获得到的预测概率去调用查询策略 `QueryStrategy`，选择出下一批需要审核的警告，然后调用 `MessageProducer` 中的方法发送需要审核的警告 ID。这是线程会陷入阻塞状态等待审核完毕，审核完毕之后再审核结果持久化更新到数据库，同时利用新得到的标签更新训练集。这时如果不满足循环条件就直接结束。如果模型需要获取项目所有警告的审核结果，也可以直接向 `LabelDao` 查询。

3.6.4 警告审核模块

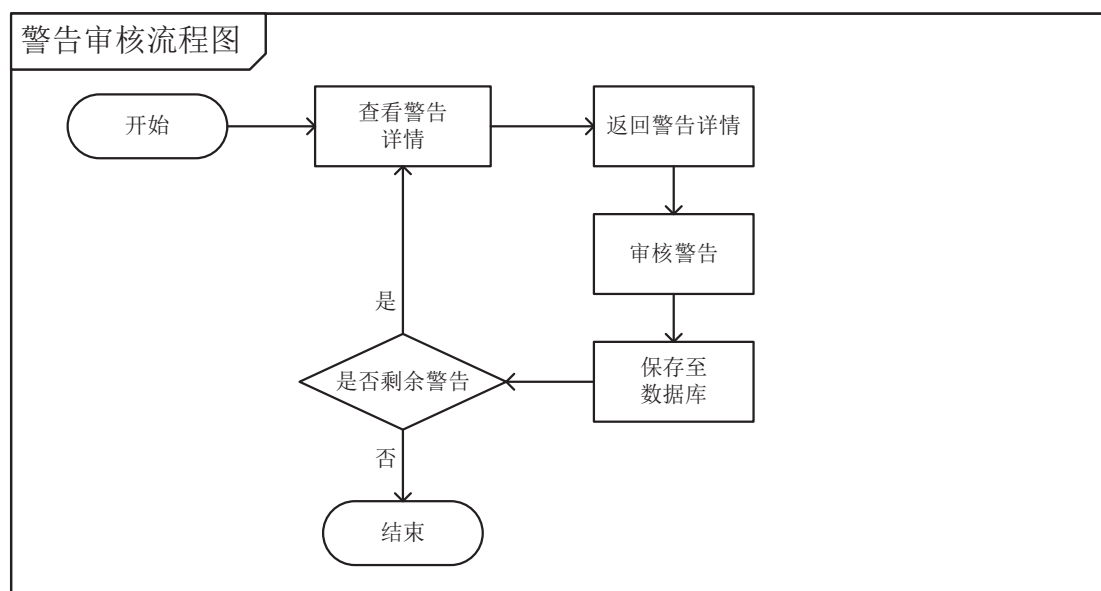


图 3.18: 警告审核模块流程图

警告审核模块负责处理警告审核相关的请求和操作，其功能相对比较简单，流程图如图 3.18 所示。审核人员先查看警告详情，这时系统会返回警告的详细细节，包括漏洞扫描工具反馈的结果、模型预测返回的概率和源代码文本。然后审核人员根据自己的专业知识做出判断，选择警告为正报或误报，并提交给系统。系统收到审核结果时及时保存到数据库。如果仍有待审核的警告，审核人员将继续审核工作流程，直到所有被分配的审核任务都已完成。

警告审核模块的核心类图如图 3.19 所示，其中 `AlarmService` 是向接口层

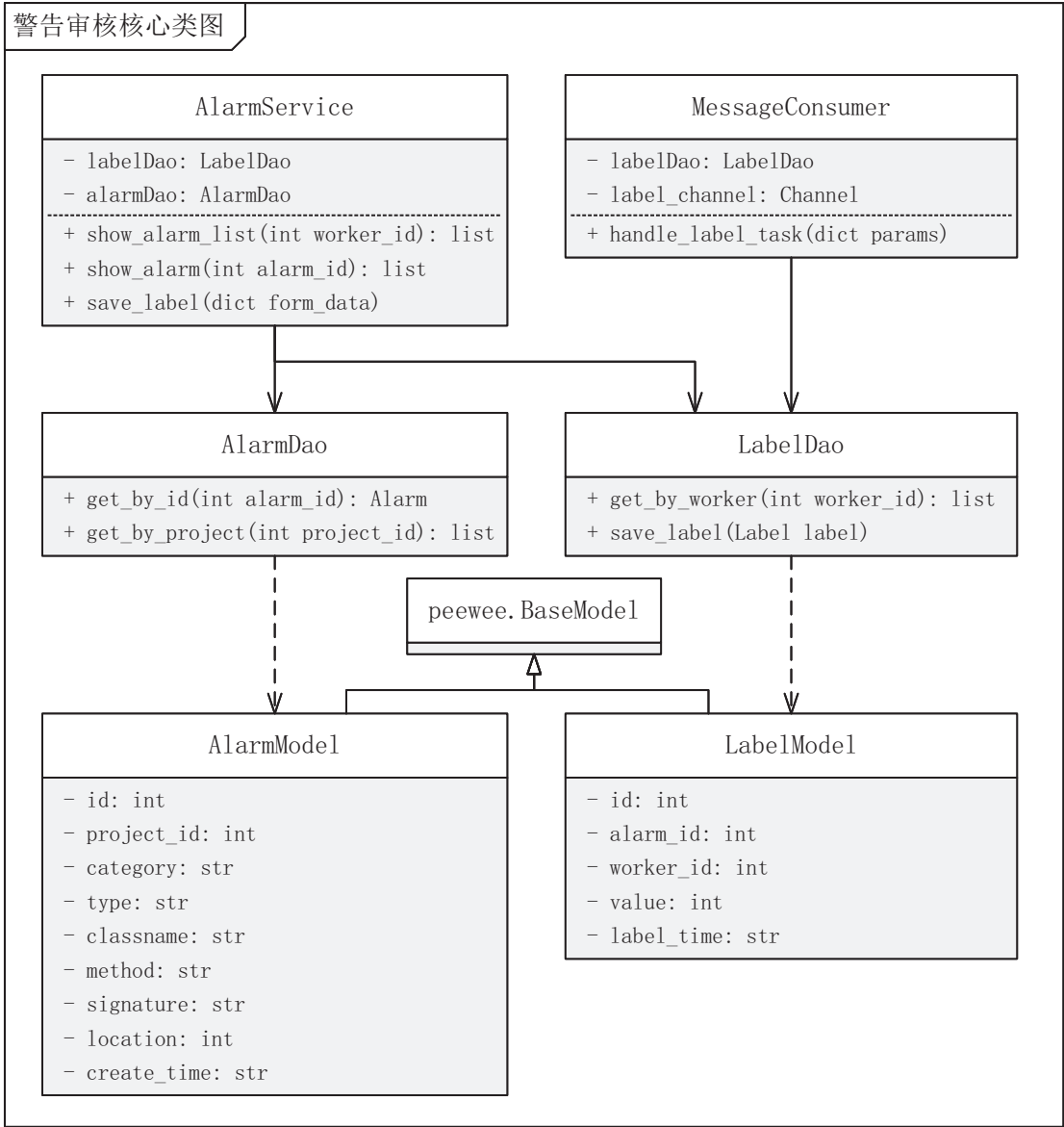


图 3.19: 警告审核模块核心类图

提供服务的服务类，会综合利用 **AlarmDao** 和 **LabelDao** 的结果组装成完整的警告信息返回给用户。其包含一个 `show_alarm_list` 方法，返回审核人员被分配的任务列表，`show_alarm` 方法返回方法的细节，`save_label` 方法保存审核结果。**AlarmDao** 和 **LabelDao** 分别利用了继承了 **BaseModel** 的 **AlarmModel** 和 **LabelModel** 实现对数据库的操作。此外还有一个 **MessageConsumer** 类负责从消息队列中取出待审核的警告 ID，并使用 **LabelDao** 将其修改为待审核状态。

本模块的时序图如 3.20 图所示，其中 **MessageConsumer** 在一开始就处于运行状态，如果从监听的消息队列中收到了审核消息，则到 **LabelDao** 中将警告

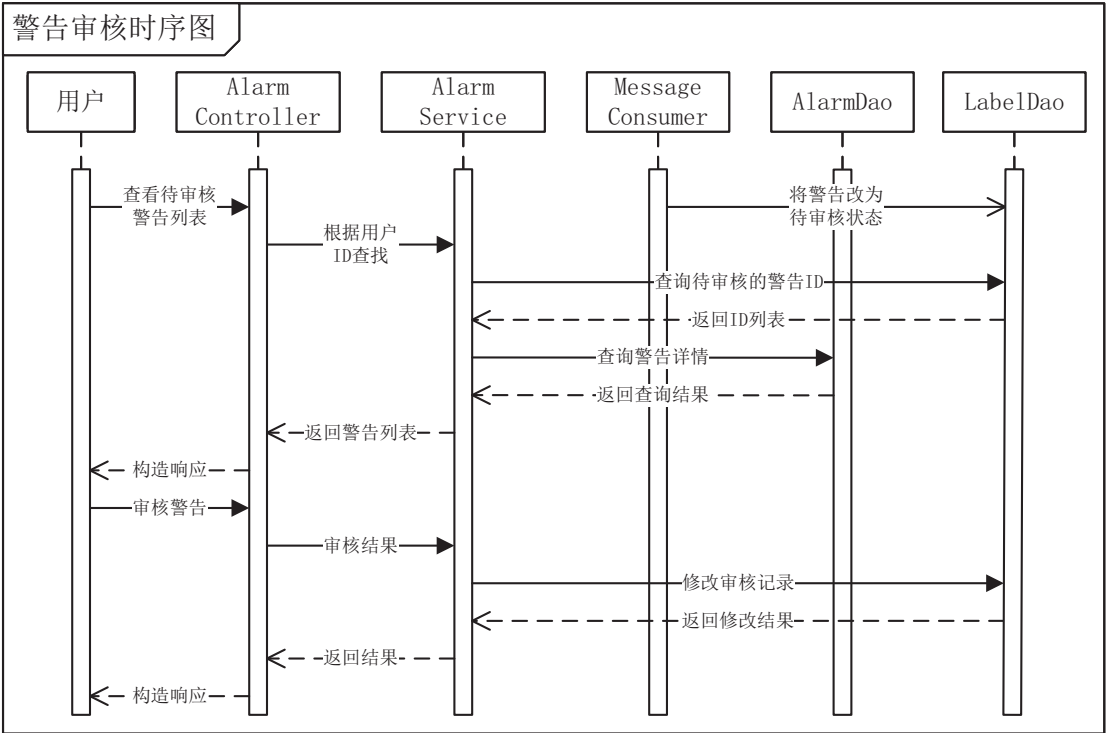


图 3.20: 警告审核模块时序图

改为待审核状态。用户先查看待审核的警告列表，AlarmController 收到请求时会从 session 中取出用户 ID，据此调用 AlarmService 中的方法查找需要审核的警告。AlarmService 会先用 LabelDao 查找分配给此审核人员的待审核的警告 ID，然后再利用 AlarmDao 查询警告的详细信息，最后组装成结果返回给 AlarmController，再由 AlarmController 构造响应返回给审核人员。审核人员提交对警告的审核之后再按顺序调用至 LabelDao 保存审核结果。

3.7 本章小结

按照软件开发的一般流程，本章详细介绍了基于主动学习的警告排序系统的需求分析与设计过程。首先，在明确系统整体使用流程的基础上，本章按照需求分析的要求依次完成了涉众分析、功能性需求分析和非功能性需求分析，并以用例图和用例描述表形成规格说明。其次，本章给出了系统的架构设计，4+1 视图和数据库设计。最后，本章将系统的主要功能划分为项目管理模块、特征提取模块、主动学习模块和警告审核模块，以流程图、核心类图和时序图描述了对各个模块的详细设计。

第四章 警告排序系统的实现

4.1 整体概述

由于本文设计的系统借助 Python 语言的 sklearn 包构建了主动学习模型，为了减少不必要的相互调用，同样使用 Python 语言构建了系统的其余主要部分，包括使用 flask 包实现了 Web 服务，peewee 包实现了数据库操作，以及使用 pika 包连接 RabbitMQ 实现消息队列的操作。只有在特征提取需要分析 Java 源代码的抽象语法树时，用到了 Java 的相关包，这部分代码会被编译打包成 Jar 包，供 Python 代码调用。

4.1.1 全局配置

系统在运行过程中需要读取或运行文件系统上的文件，为了避免将文件路径写死在代码内部，系统将这部分配置写存储在一个单独的 json 文件中。如果在系统运行中需要对这些变量进行修改，不必中断整个系统程序，只需修改 json 文件即可，增强了系统的可维护性。此外，系统在不同运行环境（例如测试环境和生产环境），或者不同操作系统（例如 Windows 和 Linux），使用的文件目录可能不一样，这时只需指定不同的 json 文件即可达到快速变更配置快速部署的效果。json 文件中保存的全局配置如表 4.1所示。

表 4.1: 系统全局配置

键名	含义	默认值
dataDir	数据父目录	"/home/root/ThesisData/"
spotBugsPath	静态代码分析工具 路径	"/home/root/spotBugs/lib/spotbugs.jar"
javaToolsPath	AST 处理工具路径	"/home/root/ThesisData/JavaTools.jar"
jhawkPath	代码度量特征提取 工具路径	"/home/root/jhawk/CommandLine/"

4.1.2 文件存储

本系统在运行时需要处理多种文件，除了用户上传的文件，还有生成的静态代码分析报告和特征数据文件。为了准确快速地查找文件，需要将文件分类存储在不同的文件目录下。因此系统规定了不同类型文件的存储目录和命名方式，所有路径均为相对于数据父目录的路径，具体约定如表 4.2 所示。

表 4.2: 系统文件存储路径

路径	说明
/project/{name}/{version}/	项目源代码文件夹
/project/{name}/{version}.jar	项目 Jar 包
/report/{name}/{version}.xml	项目缺陷检测扫描报告
/report/{name}/jhawk#{version}.xml	项目软件度量结果
/file/{name}/{version}/	格式化之后的项目源代码文件夹
/feature/{name}/{version}.csv	警告特征数据集

4.1.3 身份认证

本系统的角色身份和权限控制并不复杂，因此采用 session 存储的方式进行身份验证。HTTP 协议本身是无状态的，但是借助 cookie 和 session 的机制，客户端登录成功后，服务端就能识别其后续请求，成为有状态的协议。尽管 session 的存储需要耗费一定的服务器资源，但是考虑到本系统的并发用户数量并不多，因此使用基于 session 的登录方式可以满足实际需求。

```
@app.route("/api/login", methods=["POST"])
def login():
    # 判断是否登录
    if "id" not in session:
        username = request.form.get("username")
        password = request.form.get("password")
        # 判断密码是否正确，正确返回用户id，否则返回-1
        user_id = WorkerService.login(username, password)
        if user_id != -1:
            session["id"] = user_id
            session["username"] = username
        else:
            return '{"msg": "用户名或密码错误"}'
    else:
        return '{"msg": "用户已登录"}'
```

图 4.1: 身份认证的实现代码

登录的实现代码如图4.1所示，首先通过判断 `session` 中是否包含 `id` 字段判断登录状态，如果未登录则从提交的表单中读取用户名和密码，以 `WorkerService` 提供的接口校验。如果身份认证通过，此接口会返回该用户的 `id` 并将其和用户名保存到 `session` 中，否则返回-1 并提示用户登录失败。

为了避免未登录用户访问系统的 API 接口，需要对每个接口方法实现访问控制，借助 `Python` 的装饰器功能可以方便地完成这一需求。装饰器本身是一类特殊的函数，可以为其他函数添加新的功能，这里代码中为被修饰的函数添加访问控制的功能，即在执行被修饰的函数之前，先判断是否登录，如果没登录则跳转到登录页面。如图4.2所示系统中实现了一个名为 `login_auth` 的装饰器，在需要访问控制的函数上使用 `@login_auth` 修饰即可。需要注意的是，函数经过装饰器修饰后，其函数名会变为内部函数名 `inner`，在绑定视图端点时会出错，因此需要修改其 `__name__` 属性或者手动指定每个视图函数的端点，这里使用前者的方法。

```
def login_auth(func):
    def inner(*args, **kwargs):
        # 未登录时重定向至登录页面
        if session.get("id") is None:
            return redirect("/login")
        # 已登录则执行被包装的函数
        else:
            return func(*args, **kwargs)
    inner.__name__ == func.__name__
    return inner

# 示例
@app.route("/", methods=["GET"])
@login_auth
def main_page():
    pass
```

图 4.2: 访问控制的实现代码

4.2 项目管理模块的实现

项目管理模块的核心在于保存上传的项目并扫描警告，后续还包括警告信息的读取与持久化等操作。其余均为查询相关的操作，此处不再详述。

4.2.1 项目上传

项目文件的上传使用了 Flask 框架对表单提交文件的相关功能，普通表单的字段存储在 `request.form` 下的字典里，而文件字段是存储在 `request.files` 字典中。图 4.3 给出了项目上传的关键代码，注意仅处理 POST 请求。在使用 `save` 方法保存文件之前，要先检查其父目录是否存在，这里调用了 `PathUtils` 中的方法，如果不存在将先自动创建文件夹。为了防止保存文件的过程中出现难以预料错误，在外层使用了 `try-catch` 语句处理可能抛出的异常。

```
@app.route("/api/upload", methods=["POST"])
@login_auth
def upload_api():
    # 记录返回的提示信息
    info = ""
    # 如果是上传请求
    if request.method == "POST":
        # 省略了对参数重复性的校验
        try:
            # 接收参数
            name = request.form.get("name")
            version = request.form.get("version")
            description = request.form.get("description")
            zip_file = request.files.get("zip")
            jar_file = request.files.get("jar")
            # 文件夹不存在时创建
            project_dir = PathUtils.project_path(name)
            PathUtils.rebuild_dir(project_dir, skip=True)
            # 保存文件
            zip_file.save(PathUtils.project_path(name, version + ".zip"))
            jar_file.save(PathUtils.project_path(name, version + ".jar"))
            info = "项目上传成功"
        except Exception:
            info = "上传项目时出错"
    return '{"msg": {1}}'.format(info)
```

图 4.3: 项目上传的实现代码

4.2.2 项目扫描与处理

项目文件保存至文件系统之后会立即开始项目扫描的任务。本系统所使用的基于静态分析的缺陷检测工具是 SpotBugs^①，它是在 FindBugs^②基础上继续开发的，拥有更多的警告校验规则，并且能很好地配合 Ant、Maven、Gradle 等开发工具共同使用。系统采用命令行执行命令的方式调用 SpotBugs，其调用代码如图 4.4 所示。参数设置中，`-textui` 选项表示以命令行用户接口执行，`-low` 选项

^①<https://spotbugs.github.io/>

^②<http://findbugs.sourceforge.net/>

表示尽可能地报告所有的缺陷，-xml 选项表示输出 xml 格式的文件，-output 选项后指定输出文件的路径。

```
def run_spotbugs(jar_path: str, report_path: str):  
    """  
    使用SpotBugs扫描得到xml格式的扫描报告  
    :param jar_path: 目标Jar路径  
    :param report_path: 扫描报告保存路径  
    """  
    os.popen("java -jar {0} -textui -low -xml -output {1} {2}".  
             format(Config.SPOTBUGS_PATH, report_path, jar_path))
```

图 4.4: 调用 SpotBugs 的代码

扫描的结果保存在以 UTF-8 格式编码的 xml 文件中，根元素是 BugCollection 元素，其下面每个 BugInstance 元素表示一个警告。一个典型的 BugInstance 元素结构如图 4.5 所示，其属性中包含了类别、分类、优先级、严重程度等信息。内部通常会包含 Class 元素和 Method 元素，分别指出报出警告的类和方法的相关信息。其中的 SourceLine 元素则进一步地给出了引起警告的代码范围。系统使用 xml.etree 包中的 ElementTree 类解析 xml 文件，解析过程中注意判断 Class 和 Method 元素给出的属性是否对应，另外由于本文的方法需要较为精确的代码行，所以保留那些代码范围在同一行的警告。

```
<BugInstance type="CN_IDIOM_NO_SUPER_CALL" priority="2" rank="16" abbrev="CN"  
category="BAD_PRACTICE">  
  <Class classname="org.apache.flink.api.common.accumulators.DoubleMaximum">  
    <SourceLine  
      classname="org.apache.flink.api.common.accumulators.DoubleMaximum" start="26"  
      end="91" sourcefile="DoubleMaximum.java" sourcepath="org/apache/flink/api/common/  
      accumulators/DoubleMaximum.java"/>  
    </Class>  
    <Method classname="org.apache.flink.api.common.accumulators.DoubleMaximum"  
      name="clone" signature="()Lorg/apache/flink/api/common/accumulators/DoubleMaximum;"  
      isStatic="false">  
      <SourceLine  
        classname="org.apache.flink.api.common.accumulators.DoubleMaximum" start="68"  
        end="70" startBytecode="0" endBytecode="77" sourcefile="DoubleMaximum.java"  
        sourcepath="org/apache/flink/api/common/accumulators/DoubleMaximum.java"/>  
      </Method>  
    </BugInstance>
```

图 4.5: SpotBugs 报告示例

考虑到开发人员的编程习惯并不相同，代码中又存在着大量的注释文本，不同项目的 Java 文件源码需要进行统一的格式化处理，便于之后统一进行特征提取。但是格式化会造成引起警告的行号发生变化，因此需要找到格式化之后对应的行号。这些操作均由 Java 中的 javaparser 包实现，格式化出现警告的 Java 文件并另存至其他目录，其核心代码如图 4.6 所示。

```

public static List<Integer> formatFile(Path originalPath, Path newPath, List<Integer> lines) {
    try {
        // 开始处理
        CompilationUnit cu = StaticJavaParser.parse(originalPath);
        List<Integer> result = lines.stream()
            .map(line -> {
                // 保存目标节点
                final Node[] node = {null};
                CompilationUnit copy_cu = cu.clone();
                // 层次遍历寻找包含代码行的最小树节点
                copy_cu.walk(Node.TreeTraversal.BREADTHFIRST, n -> {
                    Range range = n.getRange().orElse(null);
                    if (range != null && node[0] == null && range.begin.line == line
                        && range.end.line == line) {
                        node[0] = n;
                    }
                });
                // 去除注释
                copy_cu.removeComment().getAllContainedComments().forEach(Comment::remove);
                // 为节点增加特有的行注释，再到之后的文本中查找
                String flagStr = Integer.toString(node[0].toString().hashCode());
                node[0].setLineComment(flagStr);
                String[] _lines = copy_cu.toString().split("\n");
                for (int i = 0; i < _lines.length; i++) {
                    if (_lines[i].endsWith(flagStr)) {
                        return i + 1;
                    }
                }
            })
        .collect(Collectors.toList());
        if (result.stream().anyMatch(i -> i != -1)) {
            cu.removeComment().getAllContainedComments().forEach(Comment::remove);
            String[] original_lines = cu.toString().split("\n");
            // 创建新文件并保存内容
            FileUtils.forceMkdirParent(newPath.toFile());
            FileUtils.touch(newPath.toFile());
            FileUtils.writelines(newPath.toFile(), Arrays.asList(original_lines));
        }
        return result;
    } catch (IOException e) {
        log.error("Parse java file error: " + originalPath);
        e.printStackTrace();
        return Collections.nCopies(-1, lines.size());
    }
}

```

图 4.6: 格式化文件并调整行号的实现代码

该方法的输入包括原来的文件路径 `originalPath`、另存的新文件路径 `newPath` 和需要调整的行号列表，调用时会以每个 Java 文件为单位，统计其中出现警告的行号列表，一并输入方法同时处理。在使用 `StaticJavaParser` 的解析方法生成抽象语法树后，对于每个行号，都以层次遍历的搜索方法找到包含这一行的代码范围最小的树节点，调整之后的行号必定在这一树节点内。接着去除所有注释后为这一行添加特殊的行注释标记，这样格式化之后注释会出现在该节点代码的同一行。利用这一特点，遍历格式化之后的代码文本，找到这一行的位置即可。最后创建新的保存目录和新文件并保存格式化之后的内容，返回调整后的行号。主程序保存警告至数据库时会用这些调整后的行号代替原有报告中的行号，之后使用数据时以修改后的文件和行号为准。

4.3 特征提取模块的实现

如第三章详细设计部分所述，特征提取模块包含四个不同方面的特征，分别在 `CodeChr`、`CodeAnl`、`WarningChr` 和 `WarningCmb` 类中实现，本节将分别介绍其具体的实现方法，以及最后对特征的合并与保存，其简单的概括说明如下：

- `CodeChr`：代码度量特征，来自于不同粒度下的软件度量指标。
- `CodeAnl`：代码分析特征，来自于对源代码 AST 的静态分析结果。
- `WarningChr`：警告自身特征，来自于静态代码分析工具的检测报告。
- `WarningCmb`：警告分布特征，来自于正报在不同警告类型上的分布情况。

4.3.1 代码度量特征

软件度量指标最初被用于量化地评价软件质量，以改进软件开发过程，其中一部分的度量是针对源代码展开的。随着机器学习技术被引入软件工程领域，代码度量特征开始被用于机器学习模型的训练中。本系统使用基于静态分析技术的 `Jhawk`^①工具获得对 Java 代码的度量结果。`Jhawk` 工具能够扫描项目文件夹下的所有 Java 源代码文件，并将度量结果以 `xml` 的格式输出成文件。本系统从其输出结果中选择了部分度量特征用作模型使用的特征中，具体如下表 4.3 所示。

本系统所使用的 `Jhawk` 工具存在着一个缺陷，无法解析包含枚举类的 Java 代码，因此本系统在分析时会跳过这些文件。这两部分的代码分别实现为 `grep_enumeration` 方法和 `run_jhawk` 方法，作用分别是找出包含枚举类的文件和调用 `Jhawk` 工具，如图 4.7 所示。

`grep` 命令是 Linux 下最常用的文本处理工具，可以快速在整个文本中搜索指定的字符串。而 Java 的枚举类是使用 `enum` 关键字定义的，因此只需递归地遍历项目文件夹下面的所有文件，过滤出包含 `enum` 的文件即可。为此使用 `-r` 选项表示递归查找，`-l` 选项表示仅输出包含匹配字符串的文件名，`-w` 选项表示匹配整个单词以免其他单词造成干扰。这里返回的文件列表会作为 `run_jhawk` 方法的 `exclude_files` 输入。

使用命令行的方式执行 `Jhawk` 是通过 Java 命令执行其目录下的 `JHawkCommandLine.jar`。这里使用 `-x` 选项指定扫描项目的路径，`-f` 选项指定扫描的文件类

^①<http://www.virtualmachinery.com/jhawkprod.htm>

表 4.3: 代码度量特征表

ID	特征名称	特征含义
F1	code_method_statement	方法中不含有注释的语句数目
F2	code_class_statement	类中不含有注释的语句数目
F3	code_package_statement	包中不含有注释的语句数目
F4	code_class_comment	类中的注释行数目
F5	code_class_comment_ratio	类中注释行长度与代码行长度的比值
F6	code_class_method	类中包含的方法数目
F7	code_package_method	包中包含的方法数目
F8	code_package_class	包中包含的类的数目
F9	code_indentation	警告行的缩进空格数目
F10	code_cyclomatic_complexity	方法内代码的圈复杂度

```
def run(command: str) -> List[str]:
    with os.popen(command) as p:
        return [s.strip() for s in p.readlines()]

def grep_enumeration(project_path: str) -> List[str]:
    return CommandUtils.run('grep -w "enum" {0} -r -l'.format(project_path))

def run_jhawk(project_path: str, report_path: str, exclude_files: List[str]):
    """
    使用JHawk命令行工具计算代码度量，并输出为xml文件
    :param project_path: 扫描项目路径
    :param report_path: 度量报告保存路径
    :param exclude_files: 排除不扫描的文件路径列表
    """
    CommandUtils.run(
        "java -jar {0}/JHawkCommandLine.jar -x {1} -f *.*.java -r -s {2} -l pcm -a"
        "-p {0}/jhawk.properties -xf \"\\Q{3}\\E\"".
        format(Config.JHAWK_PATH, report_path, project_path, "\\E|\\Q".join(exclude_files)))
```

图 4.7: 调用 Jhawk 提取代码度量特征的实现代码

型是 Java 文件，-s 选项指定了报告的保存路径，-xf 选项指定了需要跳过不扫描的文件，其余选项指定了度量指标的选择范围。跳过不扫描的文件以 \Q 和 \E 包围后再以 | 字符连接，\Q 和 \E 之间的文本如遇正则表达式中的特殊符号将自动转义，| 字符表示与其中任一文件路径匹配即跳过。

Jhawk 工具生成度量报告之后，同样使用 ElementTree 类解析 xml 文本，将相关度量值按照包名、类名、方法名的键值映射分别存入字典中。最后在需要实现的 get_feature_df 方法内部遍历警告，通过映射找出对应的度量值，作为特

征填入 DataFrame 返回。

4.3.2 代码分析特征

代码的分析特征是借助抽象语法树提取的，本系统选择的特征来自于方法和类的可见性等修饰符，如表 4.4所示。其中除返回类型为完全限定类名的字符串外，其余特征均以 1 或 0 二值化数据表示其修饰符的存在与否。

表 4.4: 代码分析特征表

ID	特征名称	特征含义
F11	method_visibility_public	方法的可见性: public
F12	method_visibility_default	方法的可见性: 默认
F13	method_visibility_protected	方法的可见性: protected
F14	method_visibility_private	方法的可见性: private
F15	return_type	方法的返回类型
F16	method_static	方法的修饰符: static
F17	method_final	方法的修饰符: final
F18	method_abstract	方法的修饰符: abstract
F19	method_protected	方法的修饰符: protected
F20	class_visibility_public	类的可见性: public
F21	class_visibility_default	类的可见性: 默认
F22	class_visibility_protected	类的可见性: protected
F23	class_visibility_private	类的可见性: private
F24	class_abstract	类的类别: 抽象类
F25	class_interface	类的类别: 接口
F26	class_enum	类的类别: 枚举类

本系统会遍历代码目录下的 java 文件，并对每个文件应用图 4.8中的 accept 方法。具体思路为分别找到文件中所有的类、接口、枚举类和构造方法，应用不同的方法提取信息。提取信息的方法是 handleClass 和 handleMethod，其作用为遍历所找的节点的修饰符，对出现的特征标为 1，其余默认为 0。接着将类或方法的完全限定名与特征信息的对应关系以键值对的形式保存在 classMap 和 methodMap 中。这里注意构造方法的完全限定名和扫描报告的命名方式保持一致，均为 <init> 字符串。最后将两个 Map 输出为 JSON 字符串，Python 端程序在调用完成之后就可以从输出流中提取字符串并解码成 python 内部的字典数据

```

private void accept(Path p) {
    try {
        CompilationUnit cu = StaticJavaParser.parse(p);
        // 处理类和接口
        cu.findAll(ClassOrInterfaceDeclaration.class)
            .forEach(d -> d.getFullyQualifiedName().ifPresent(className -> {
                ClassInfo info = handleClass(d);
                info.setInterface(d.isInterface());
                classMap.put(className, info);
            }));
        // 处理枚举类
        cu.findAll(EnumDeclaration.class)
            .forEach(d -> d.getFullyQualifiedName().ifPresent(className -> {
                ClassInfo info = handleClass(d);
                info.setEnum(true);
                classMap.put(className, info);
            }));
        // 处理方法
        cu.findAll(MethodDeclaration.class)
            .forEach(m -> {
                String className = findParentClass(m).getFullyQualifiedName()
                    .orElse("");
                String methodName = m.getNameAsString();
                MethodInfo info = handleMethod(m);
                info.setReturnType(m.getTypeAsString());
                methodMap.put(className + "." + methodName, info);
            });
        // 处理构造方法
        cu.findAll(ConstructorDeclaration.class)
            .forEach(m -> {
                String className = findParentClass(m).getFullyQualifiedName()
                    .orElse("");
                methodMap.put(className + "<init>", handleMethod(m));
            });
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

图 4.8: 代码分析特征提取的实现代码

类型，以此达到数据结果传输的效果。

4.3.3 警告自身特征

警告自身的特征主要来自于静态代码分析工具的扫描报告中给出的数据。在此基础上，基于警告可能会聚集在一起的分布特点，本系统还进一步统计了不同粒度下警告的数目作为特征。对警告自身特征的详细介绍如表 4.5 所示。

F27-F29 的几个特征可以直接从警告 BugInstance 元素的 type、priority 和 rank 字段读取。F30-F32 的几个特征需要统计不同粒度下的警告数目，对此新增一列作为分类的键值然后组合使用 groupby 和 count 方法进行分类汇总。其具体方法如图 4.9 所示。可以看到，以方法为粒度是按类名和方法名的组合分类汇

表 4.5: 警告自身特征表

ID	特征名称	特征含义
F27	warning_type	警告的类型（缺陷模式）
F28	warning_priority	警告的优先度
F29	warning_rank	警告的威胁等级
F30	warning_num_method	同一个方法中的警告数目
F31	warning_num_class	同一个类中的警告数目
F32	warning_num_package	同一个包中的警告数目

```
class WarningChr(FeatureCategory):
    def __init__(self, alarm_df: pd.DataFrame, project_name: str, version: str):
        super().__init__(alarm_df, project_name, version)

    def get_feature_df(self) -> pd.DataFrame:
        columns = [self._warning_type(), self._warning_priority(),
                  self._warning_rank(), self._num_in_method(),
                  self._num_in_class(), self._num_in_package()]
        return pd.DataFrame({s.name: s for s in columns}, index=self.alarm_df.index)

    def _num_in_method(self) -> pd.Series:
        method_num = self.alarm_df.groupby(["class_name", "method"])["version"].count()
        series = self.alarm_df
        series.apply(lambda row: method_num[row["class_name"], row["method"]], axis=1)
        series.name = FeatureType.F94
        return series

    def _num_in_class(self) -> pd.Series:
        # 内部类以外部类为准，取分隔符$前字符串
        self.alarm_df["class"] = self.alarm_df["class_name"].map(lambda s: s.split("$")[0])
        class_num = self.alarm_df.groupby("class")["version"].count()
        series = self.alarm_df["class"].map(lambda s: class_num[s])
        series.name = FeatureType.F95
        return series

    def _num_in_package(self) -> pd.Series:
        # 取最后一个.号前面的字符串作为包名
        self.alarm_df["package"] = self.alarm_df["class_name"]
        self.alarm_df["package"] = self.alarm_df["package"].map(lambda s: ".".join(s.split(".")[:-1]))
        package_num = self.alarm_df.groupby("package")["version"].count()
        series = self.alarm_df["package"].map(lambda s: package_num[s])
        series.name = FeatureType.F96
        return series
```

图 4.9: 警告自身特征提取的实现代码

总，以类为粒度是按外部类的完全限定名分类汇总，而包粒度则是去类名最后一个.号前面的字符串作为包名分类汇总。汇总结束后再对每个警告构造相应的键值，到汇总结果中取出对应的特征值。这三个方法返回的 Series 将作为特征列合并到结果当中。

4.3.4 警告分布特征

警告分布特征来自于对历史数据中同一类警告下是正报或误报的数目统计，在此基础上处理成分布概率等形式，具体如表 4.6 所示。

表 4.6: 警告分布特征表

ID	特征名称	特征含义
F33	warning_pattern_likelihood	此模式中的警告是误报的概率
F34	warning_likelihood_variance	F33 概率的方差
F35	warning_type_likelihood	此类型中的警告是误报的概率
F36	warning_discretization	F33 概率的离散化程度

下面介绍这几个特征的计算方法。SpotBugs 的文档中给出了对不同缺陷模式的描述^①。从小标题中可以看出不同的缺陷模式有不同的缩写，第一个下划线之前的内容可看作是模式的分类。对此本文假设存在着多个不同的缺陷分类 $K_1, K_2, \dots, K_m$ 。对于任意一个缺陷分类 K_i ，包含多个缺陷模式 $P_{i1}, P_{i2}, \dots, P_{in}$ 。对于任一个缺陷模式 P_{ij} ，其中警告为误报的数量记为 P_{ijF} ，警告为正报的数量记为 P_{ijT} ，且有 $P_{ij} = P_{ijF} + P_{ijT}$ 。在此基础上，本文利用 (4-1) 中的公式计算模式中的警告是误报的概率。

$$D(P_{ij}) = \frac{P_{ijF}}{P_{ijF} + P_{ijT}} \quad (4-1)$$

由于不同缺陷分类包含的缺陷模式数目不一，为避免其对误报可能性计算的影响，本文使用 (4-2) 中的公式计算方差 $V(P_{ij})$ 作为另一个指标。

$$V(P_{ij}) = \frac{D(P_{ij}) \times (1 - D(P_{ij}))}{n} \quad (4-2)$$

假设有两个缺陷模式具有相同的误报概率，但是给出警告更多的那个缺陷模式的方差值更小，也就是意味着它的变化程度更低，可能先检查其对应的警告更为合适。得到每个缺陷模式的误报概率之后可以继续使用公式 (4-3) 计算此分类的误报概率。

$$D(K_i) = \frac{\sum_1^n P_{ij} \times D(P_{ij})}{\sum_1^n P_{ij}} \quad (4-3)$$

^①<https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html>

这一指标忽略了正误报在分类下各模式中的分布。例如，两个缺陷分类 K_1 和 K_2 ，分别包含两个缺陷模式， K_1 中两个模式各有 5 个误报和 5 个正报，因此计算得到 $D(K_1)$ 为 50%。而 K_2 中一个模式有 10 个正报和 0 个误报，另一个模式恰好相反，有 0 个正报和 10 个误报，此时 $D(K_2)$ 也为 50%，但显然先检查 K_1 更合适，因为其误报的分布更加集中，下面 (4-4) 中的公式用于计算缺陷分类的离散化程度 S_i 。

$$S_i^2 = \frac{1}{n-1} \sum_{j=1}^n (D(P_{ij}) - D(K_i))^2 \quad (4-4)$$

4.3.5 特征合并

本系统处理特征是以项目的每个版本为单位，一次性提取所有其下所有警告的特征。本系统使用如图 4.10 所示的代码分别提取特征，每个实现了 FeatureExtraction 的类在构造之后会被放入 extractors 列表中，之后依次调用 get_feature_df 方法获得特征数据并使用 join 方法合并至已有数据。

```
def extract(alarm_df: DataFrame, project_name: str, version: str) -> DataFrame:
    """
    提取一个版本对应警告的特征
    :param alarm_df: 警告DataFrame
    :param project_name: 项目名称
    :param version: 版本
    :return: 特征DataFrame
    """
    feature_df = pd.DataFrame(index=group_df.index)
    # 分别提取特征
    extractors = []
    extractors.append(CodeAnl(alarm_df, project_name, version))
    extractors.append(CodeChr(alarm_df, project_name, version))
    extractors.append(WarningChr(alarm_df, project_name, version))
    extractors.append(WarningCmb(alarm_df, project_name, version))
    # 合并不同类的特征DataFrame
    for extractor in extractors:
        feature_df = feature_df.join(extractor.get_feature_df())
    return feature_df
```

图 4.10: 特征合并的实现代码

4.4 主动学习模块的实现

和详细设计中略有不同的是，在使用主动学习模型之前，系统会先查找和上一版本中相同的被标记为正报的警告。这些警告在当前版本也很有可能是正

报，所以先送给审核人员进行标记审核的工作。在机器学习模型部分，考虑到误报数据要远远多于正报数据这一不平衡的情况，代码中修改了 `sklearn` 原有的集成学习分类器，使其在真正进行训练之前调整了训练集的平衡性。

4.4.1 相同警告预排序

本系统采用三种不同的方法判断两个警告是否相同，分别是基于位置的方法、基于片段的方法和基于哈希值的方法。在两个版本的所有警告之间成对地判断是否相同会耗费大量的时间。因此总体上的判断逻辑是，先以文件为单位，采用基于位置和片段的方法两两比较相同文件中的警告。再其次考虑到代码在不同版本之间可能会有重构的情况发生，就不会被前面的方法所匹配，因此在这种情况下再使用基于哈希值的方法，将警告哈希值与前一版本的所有警告的哈希值相比，如果有相同的哈希值，则可以认为是找到了一组相同的警告。

首先介绍基于位置的判断方法。该方法使用 `difflib` 包中提供的算法计算两个给定文件 F_a 和 F_b 的差异对，将每个文件分成同样数目的 n 个区间，分别记为 $r_1^a, r_2^a, \dots, r_n^a$ 和 $r_1^b, r_2^b, \dots, r_n^b$ 。也就是说一个相同序号的区间对 (r_i^a, r_i^b) 是一个匹配对或者差异对。考虑到实际开发中 `java` 文件的开头存在着一行 `package` 关键字的包管理语句，且匹配对和差异对是间隔出现，因此序号为奇数的区间必然为匹配对，而序号为偶数的区间必然为差异对。

对于警告中提供的行号 l ，使用 $[l]$ 表示行号所在区间的起始行，用 $\Delta(l)$ 表示行号到起始行的距离。那么对于两个警告所给出的代码行位置 l_a 和 l_b ，如果在同一区间对中，分两种情况：

- 如果在同一匹配对之中，只有位置完全相同，即 $\Delta(l_a) = \Delta(l_b)$ 时认为它们是相同的警告。
- 如果在同一差异对之中，它们的位置可以有一些偏离，因此只要位置差值在一定阈值范围之内，即 $|\Delta(l_a) - \Delta(l_b)| \leq \varepsilon$ ，就可以认为是相同的警告。本文将这个阈值设为 3。

该方法的实现代码如图 4.11 所示。`location_match` 方法将一个警告与其下一个版本中同一文件内报告的所有警告相比较，最终输出匹配的警告，如果没有满足匹配条件的警告，则返回 `None`。由于警告都来自于同一个文件，所以确定匹配警告的行号范围只需要计算一次算法。首先使用 `difflib` 的 `unified_diff`

```

def location_match(self, alarm: Alarm, alarm_group: List[Alarm]) -> Alarm:
    """
    同一文件内，相同分类的警告匹配，如果存在多个，返回最匹配的
    :param alarm: 当前版本的警告
    :param alarm_group: 下一版本中同文件同类型的警告分组，默认非空
    :return: 下一版本中最匹配的警告，如果没有则返回None
    """
    # 最匹配的警告
    matched_alarm = None
    # 前后版本的文件路径
    file_path_a = PathUtils.file_path(self.project_name, alarm.version, alarm.path)
    file_path_b = PathUtils.file_path(self.project_name, alarm_group[0].version, alarm_group[0].path)
    # 前后版本的文件内容
    file_content_a = open(file_path_a, "r").readlines()
    file_content_b = open(file_path_b, "r").readlines()
    # 标记了行号范围，必定从匹配块开始
    line_range_a = [0]
    line_range_b = [0]
    # 比较文件的差异并进行分块
    for line in difflib.unified_diff(file_content_a, file_content_b, n=0):
        reg = re.match(r"@@ -(\d*),(\d*) \+(\d*),(\d*) @@", line)
        if reg is not None:
            line_range_a.extend([int(reg.group(1)), int(reg.group(1)) + int(reg.group(2))])
            line_range_b.extend([int(reg.group(3)), int(reg.group(3)) + int(reg.group(4))])
        elif reg := re.match(r"@@ -(\d*) \+(\d*) @@", line):
            line_range_a.extend([int(reg.group(1)), int(reg.group(1)) + 1])
            line_range_b.extend([int(reg.group(2)), int(reg.group(2)) + 1])
        elif reg := re.match(r"@@ -(\d*),(\d*) \+(\d*) @@", line):
            line_range_a.extend([int(reg.group(1)), int(reg.group(1)) + int(reg.group(2))])
            line_range_b.extend([int(reg.group(3)), int(reg.group(3)) + 1])
        elif reg := re.match(r"@@ -(\d*) \+(\d*),(\d*) @@", line):
            line_range_a.extend([int(reg.group(1)), int(reg.group(1)) + 1])
            line_range_b.extend([int(reg.group(2)), int(reg.group(2)) + int(reg.group(3))])
    line_range_a.append(len(file_content_a))
    line_range_b.append(len(file_content_b))
    # 判断是匹配块还是差异块
    index = 0
    while line_range_a[index + 1] <= alarm.new_location:
        index += 1
    # 如果是匹配块
    if index % 2 == 0:
        # 找一个位置差相同的警告
        target_location = line_range_b[index] + (alarm.new_location - line_range_a[index])
        for alarm_b in alarm_group:
            if alarm_b.new_location == target_location:
                matched_alarm = alarm_b
    # 如果是差异块
    else:
        delta_a = alarm.new_location - line_range_a[index]
        alarms_in_range = list(filter(
            lambda a: line_range_b[index] <= a.new_location < line_range_b[index + 1], alarm_group))
        alarms_in_range = list(filter(
            lambda a: abs(a.new_location - line_range_b[index] - delta_a) <= self.location_delta,
            alarms_in_range))
        alarms_in_range.sort(key=lambda a: abs(a.new_location - line_range_b[index] - delta_a))
        if len(alarms_in_range) > 0:
            matched_alarm = alarms_in_range[0]
    return matched_alarm

```

图 4.11: 基于位置匹配的实现代码

方法获得差异算法的计算结果，其中存在差异的行号范围会以“@@ -3,5 +3,7 @@”类似的标记行体现出来，因此使用正则表达式去捕获行号的范围，相邻的行号范围即组成了上文所说的区间。接下来根据区间对是差异对或匹配对来计算匹配警告应当报告的行号（或行号的范围）。最后在传入的警告列表中找到最符合匹配条件的警告。

基于位置的匹配不能应对在同一文件内大范围移动代码的情形，因此还使

用了另一种基于片段的匹配方法。基于片段的方法直接比较代码行范围的内容，如果完全相同则可以认为是匹配警告。

```
def _get_hash_token(self, alarm: Alarm) -> Tuple[List[str], List[str]]:
    """
    获取警告位置前后的token序列
    :param alarm:
    :return: 如果不足返回None
    """
    # 读取相应文件
    file_path = PathUtils.file_path(self.project_name, alarm.version, alarm.path)
    file_content = open(file_path, "r").readlines()
    # 分词得到token并去除空字符串
    token_lines = [re.split(r"[\{\};\+\*\[\]\.\.\|\(\)\?^\-/:&<>\s]", line) for line in
file_content]
    token_lines = [list(filter(lambda w: w != "", line)) for line in token_lines]
    # 警告位置之前的token
    token_before = []
    for i in range(alarm.new_location - 2, -1, -1):
        if len(token_before) + len(token_lines[i]) <= self.hash_delta:
            token_before = token_lines[i] + token_before
        else:
            token_before = token_lines[i][len(token_before) - self.hash_delta:] + token_before
    # 警告位置之后的token
    token_after = []
    for i in range(alarm.new_location - 1, len(token_lines)):
        if len(token_after) + len(token_lines[i]) <= self.hash_delta:
            token_after = token_after + token_lines[i]
        else:
            token_after = token_after + token_lines[i][:len(token_after) - self.hash_delta]
    if len(token_before) < self.hash_delta:
        token_before = None
    if len(token_after) < self.hash_delta:
        token_after = None
    return token_before, token_after
```

图 4.12: 基于哈希值匹配的实现代码

不同文件中的比较采用基于哈希值的匹配方法。假设代码在不同文件中移动时，报告警告的代码行前后不会发生变化，这会使得同样的警告得以出现在另一个文件中。基于这一假设，代码会分别比较代码行以及前后代码的 token 序列的哈希值。具体来说，使用 h_1 和 h_2 分别表示警告代码前后 token 的哈希值， h_1 使用的 token 序列是从代码行第一个 token 以及其之前的共计 n 个 token， h_2 使用的 token 序列是第一个 token 及其之后的共计 n 个 token。如果两个警告的 h_1 和 h_2 中有任意一个相等，则认为它们是匹配的。这里将 n 的值设为 100，如果用于计算哈希值的 token 序列长度不足 n ，则不参与哈希值的比较。基于哈希匹配的实现代码如图 4.12 所示，首先对源代码进行分词处理，即去除所有特殊符号并删除空字符串获得单词序列，然后分别取前面和后面的 token，如果前后的序列长度小于需要的值，则返回 None。

4.4.2 机器学习模型

在相关技术综述一章，本文介绍了三种主流的集成学习模型，分别是 Bagging 方法、Boosting 方法和 Stacking 方法。本系统在方法实现时，部分地修改了其中 Bagging 方法和 Stacking 方法的流程，使其适应正误报数据集不平衡的情况，即通常情况下正报的数量会远远小于误报的数量。下面详细描述这一改进方法。

```
class BaggingClassifierModel(ModelBase):

    def __init__(self):
        super().__init__("bagging")
        self.model = BaggingClassifier(bootstrap=False, n_estimators=20,
max_samples=0.1)

    def train(self, x_data: np.ndarray, x_label: np.ndarray):
        _original_function = bagging._parallel_build_estimators
        bagging._parallel_build_estimators = modified_parallel_build_estimators
        self.model.fit(x_data, x_label)
        bagging._parallel_build_estimators = _original_function

    def predict_label(self, x_data: np.ndarray) -> np.ndarray:
        return self.model.predict(x_data)

    def predict_prob(self, x_data: np.ndarray) -> np.ndarray:
        return np.ones(x_data.shape[0]) - self.model.predict_proba(x_data)[:, 0]
```

图 4.13: BaggingClassifierModel 的实现代码

原来的 BaggingClassifier 从原始训练集中采用有放回的抽样方法，这里被改为从少数类与多数类中分别抽样。如果少数类样本数目充足，则按照近似 1:1 的比例从少数类与多数类中分别抽样。如果少数类的数目不足，就优先选择所有的少数类样本，不够的从多数类中抽样补足。整个类的实现如图 4.13 所示，修改后的方法被实现为 `modified_parallel_build_estimators`，用于在训练开始前替换原始的方法。

而原有的 StackingClassifier 方法是使用不同的弱学习器在相同的训练集上训练，这里被修改为当训练集样本数充足时，按近似 1:1 的比例从少数类与多数类中分别抽样，再进行训练。整个类的实现如图 4.14 所示，这一方法被实现为 `modified_fit_single_estimator`。本系统所使用的弱学习器包括支持向量机、决策树、朴素贝叶斯和多层感知器，最后使用 Logistic 回归得到最终分类结果。这里返回预测概率时用 `decision_function` 的返回结果。

除了集成学习器之外系统中也实现了传统的多分类机器学习模型，但是这部分没有加入数据集平衡的操作。其实现代码如图 4.15 所示，根据传入名称的

```

class StackingClassifierModel(ModelBase):

    def __init__(self):
        super().__init__("stacking")
        s._fit_single_estimator = modified_fit_single_estimator
        self.model = s.StackingClassifier(list({
            "svm": CalibratedClassifierCV(base_estimator=LinearSVC()),
            "dt": DecisionTreeClassifier(),
            "nb": GaussianNB(),
            "mlp": MLPClassifier(max_iter=1000)
        }).items()), final_estimator=LogisticRegression())

    def train(self, x_data: np.ndarray, x_label: np.ndarray):
        self.model.fit(x_data, x_label)
        print(self.model.score(x_data, x_label))

    def predict_label(self, x_data: np.ndarray) -> np.ndarray:
        return self.model.predict(x_data)

    def predict_prob(self, x_data: np.ndarray) -> np.ndarray:
        return np.abs(self.model.decision_function(x_data))

```

图 4.14: StackingClassifierModel 的实现代码

```

class MultiplyClassifierModel(ModelBase):

    def __init__(self, name: str):
        model_dict = {
            "svm": SVC(C=0.8, max_iter=2000, tol=0.001, probability=True),
            "dt": DecisionTreeClassifier(),
            "nb": GaussianNB(),
            "mlp": MLPClassifier(max_iter=1000)
        }
        # 默认的分类器
        if name not in model_dict.keys():
            name = "svm"
        self.model = model_dict[name]
        super().__init__(name)

    def train(self, x_data: np.ndarray, x_label: np.ndarray):
        self.model.fit(x_data, x_label)

    def predict_label(self, x_data: np.ndarray) -> np.ndarray:
        return self.model.predict(x_data)

    def predict_prob(self, x_data: np.ndarray) -> np.ndarray:
        prob_result = self.model.predict_proba(x_data)
        return np.apply_along_axis(np.max, 1, prob_result)

```

图 4.15: MultiplyClassifierModel 的实现代码

不同构造不同的机器学习模型，包括支持向量机、决策树、朴素贝叶斯和多层感知器，复用了模型训练和概率预测部分的代码。

4.4.3 查询策略

主动学习和传统的机器学习方法之间的主要区别在于能根据历史查询和训练的结果来查询实例。系统所实现的查询策略包括确定性采样和不确定

```

class UncertainlyQuery(QueryBase):
    """不确定性采样查询策略"""

    def query(self, labeled_index_set: set, predict_prob: pd.Series) -> pd.Index:
        unlabeled_index_set = set(self.data_df.index.to_list()) - labeled_index_set
        unlabeled_prob = predict_prob[unlabeled_index_set]
        unlabeled_prob.sort_values(inplace=True, key=lambda p: abs(p - 0.5))
        if len(unlabeled_prob) > self.max_num:
            return unlabeled_prob.iloc[:self.max_num].index
        else:
            return unlabeled_prob.index

class CertainlyQuery(QueryBase):
    """确定性采样查询策略"""

    def query(self, labeled_index_set: set, predict_prob: pd.Series) -> pd.Index:
        unlabeled_index_set = set(self.data_df.index.to_list()) - labeled_index_set
        unlabeled_prob = predict_prob[unlabeled_index_set]
        unlabeled_prob.sort_values(inplace=True, key=lambda p: - abs(p - 0.5))
        if len(unlabeled_prob) > self.max_num:
            return unlabeled_prob.iloc[:self.max_num].index
        else:
            return unlabeled_prob.index

```

图 4.16: MultiplyClassifierModel 的实现代码

性采样两种，如图 4.16 所示。调用查询方法时需要输入所有实例的预测概率 `predict_prob` 和已标记的实例索引集合 `labeled_index_set`。首先使用集合求差集的方法得到未标记的实例集合，然后取出它们的预测概率排序，确定性采样选择那些概率最接近 0 或 1 的实例，而不确定性采样选择那些概率在 0.5 附近的实例。最后根据设定的采样数目返回选择需要查询的样本实例索引。

4.4.4 整体流程

主动学习流程的核心代码如图 4.17 所示。首先，输入所有警告特征数据后，清洗掉含有空值的数据行，并临时新增一列用于记录每轮训练时模型对数据标签的预测结果。然后调用私有的 `build` 方法构造或更新实验各流程组件模块，注意这里学习模型可以跨版本使用，所以只在第一次训练时构造模型，实验的查询采样记录也在这时更新。后面先根据模型是否被训练过判断是否为第一个版本的训练，如果没有被训练过会抛出异常，从而执行初始化采样的流程，否则根据模型的预测结果取出最有可能是正报的样本送给人工审核。这里注意为保证之后的模型训练时有两个不同分类的标签样本，需要向后一直审核到两个分类的样本均存在。

这时系统得到了两个数据集索引集合，`unlabeled_index_set` 表示未标记的数

```

def run(self, data_df: pd.DataFrame):
    # 更新训练数据集
    self.data_df = data_df.copy()
    self.data_df["model_label"] = pd.Series(Alarm.UNKNOWN, index=self.data_df.index)
    self.data_df = self.data_df.dropna()
    # 更新实验流程组件, 仅在第一次训练时更新模型
    self.init_sample = self._build_init_sample()
    if self.learn_model is None:
        self.learn_model = self._build_learning_model()
    self.query_strategy = self._build_query_strategy()
    self.stop_strategy = self._build_stop_strategy()
    self.sample_records.clear()
    # 如果是初次训练则使用初始化采样, 否则选择已有的模型预测
    try:
        check_is_fitted(self.learn_model.model)
        x_data, y_label, _ = self.data_handler.preprocess(self.data_df, True)
        prob_series = pd.Series(self.learn_model.predict_prob(x_data), self.data_df.index)
        prob_series.sort_values(inplace=True, ascending=True)
        # 向后标记直到0, 1均有
        for i in range(1, len(prob_series)):
            if 1 < self.data_df.loc[prob_series.iloc[:i + 1].index, "label"].sum() < i:
                labeled_index_set = set(prob_series.iloc[:i + 1].index)
                break
        else:
            labeled_index_set = set(prob_series.index)
    except NotFittedError:
        # 初始化采样标记一部分数据
        labeled_index_set = set(self.init_sample.get_sample_index(self.data_df))
    # 剩下的为未标记数据
    unlabeled_index_set = set(self.data_df.index.tolist()) - labeled_index_set
    # 查询标签
    self._query_labels(labeled_index_set)
    # 开始主动学习步骤
    while len(unlabeled_index_set) > 0:
        # 根据停止策略决定是否终止主动学习循环
        if self.stop_strategy.judge_stop(self.data_df["label"], self.data_df["model_label"]):
            break
        # 获取训练集和标签
        train_df = self.data_df.loc[labeled_index_set]
        train_data, train_label, _ = self.data_handler.preprocess(train_df, True)
        # 模型训练
        self.learn_model.train(train_data, train_label)
        # 计算所有样本的预测结果
        pre_data, y_true, _ = self.data_handler.preprocess(self.data_df, False)
        self.data_df["label"] = pd.Series(self.learn_model.predict_label(pre_data), self.data_df.index)
        prob_series = pd.Series(self.learn_model.predict_prob(pre_data), self.data_df.index)
        # 使用查询策略得到下一批标记的数据
        label_index = self.query_strategy.query(labeled_index_set, prob_series)
        labeled_index_set = labeled_index_set | set(label_index.to_list())
        unlabeled_index_set = unlabeled_index_set - set(label_index.to_list())
        # 查询样本的标签值
        self._query_labels(set(label_index.to_list()))
        # 剩余标签不足以再次查询
        if len(unlabeled_index_set) < self.config["query_strategy"]["max_num"]:
            break

```

图 4.17: 主动学习模型的实现代码

数据集索引, `labeled_index_set` 表示已标记的数据集索引。接下来进入主动学习的循环过程, 每次循环必须满足的前置条件是, 仍有未标记数据存在并且不满足停止策略所设定的停止条件。在每个循环中, 首先根据已标记数据的索引取出数据, 预处理之后送给模型训练。接着使用训练过的模型对所有样本进行标签预测, 并将预测结果和已标记的数据索引送入查询策略中, 查询策略组件会决定下一批应当查询标签的数据索引, 并使用 `_query_labels` 方法查询。获取查询结果后会进入下一轮循环。

4.5 警告审核模块的实现

警告审核模块的核心代码是通过消息队列接收并分配审核任务，其实现代码如图 4.18 所示。

```
import random
from pika import BlockingConnection, ConnectionParameters, PlainCredentials
from model import WorkerModel, LabelModel

# 创建连接
user_key = PlainCredentials("root","admin12345")
conn = BlockingConnection(ConnectionParameters("127.0.0.1", 5672,
credentials=user_key))

# 声明管道
label_channel = conn.channel()

# 声明队列
label_channel.queue_declare(queue="label")

# 消息处理方法
def label_callback(channel, method, properties, body):
    worker_id_list = WorkerModel.select(WorkerModel.id).distinct()
    worker_id = random.sample(worker_id_list, 1)
    LabelModel.create(alarm_id=body, worker_id=worker_id, value=-1)

# 准备接收
label_channel.basic_consume(label_callback, queue="label", no_ack=True)

# 开始接收
label_channel.start_consuming()
```

图 4.18: 消息队列接收审核任务的实现代码

首先根据服务器上安装并配置好的 RabbitMQ 服务器用户名和密码创建身份令牌 `user_key`，并在创建连接时传入。接着声明了一个管道 `label_channel`，并在此管道中声明一个名为 `label` 的队列。然后系统实现了消息处理方法 `label_callback`，在收到消息时从审核人员列表中随机选择一位审核人员分配审核任务，以消息中的警告 ID 和审核人员 ID 创建一条审核记录，此时未经审核的标签值为-1。

4.6 本章小结

本章介绍了四个主要模块的具体实现过程，即项目管理模块、特征提取模块、主动学习模块和警告审核模块，着重描述了各模块核心代码的工作逻辑和实现依据。此外对系统的全局设置也做了一定的介绍。

第五章 实验设计与系统测试

5.1 实验设计

为了对本文提出的基于主动学习的警告误报检测算法进行验证，本章利用从 Maven 上搜集的部分项目，在对其进行了警告扫描和正误报标记的工作后，构建起警告误报数据集。在这一数据集的基础上，本章设计了两个实验，验证本文方法的有效性。

5.1.1 研究问题

为了验证本文提出的基于主动学习的警告排序优化方法的有效性，通过后文进行的实验，将对下列研究问题进行探索：

- RQ1: 和其他警告排序方法相比，本文提出的基于主动学习的警告排序方法是否更有效？
- RQ2: 主动学习流程中，应用不同的机器学习模型对排序效果有何影响？

RQ1 中，本文参考了部分以往的研究，设计了 5 种不同的排序方法作为对照，具体方法如下：

BaselineA

随机排序方法，即将所有警告随机排序后获得其标记结果。

BaselineB

按照静态代码分析工具 Spotbugs 给出的 rank 大小排序。

BaselineC

Heckman 等人 [45] 的研究中使用警告类型准确性 (ATA) 和代码局部性 (CL) 作为警告排序因子。对于某一类警告群体 p ，记其中正报的数量为 T_p ，误报的数量为 F_p 。这里的 p 表示一种分类的类别，例如 type 表示按警告类型分类，class 表示按警告所在类名分类，等等。对于任一警告 a ，首先计算它所属的警告群体 p 占警告总体 $total$ 的占比 t_p (5-1)：

$$t_p = \frac{T_p + F_p}{total} \quad (5-1)$$

然后计算与开发人员对警告的反馈相关的调节因子 AF_p (5-2)：

$$AF_p = \frac{F_p - T_p}{F_p + T_p} \quad (5-2)$$

最后用下列公式分别计算 $ATA(a)$ 和 $CL(a)$ ，其中下标 p 为 type、sf、c、m 时分别表示按警告类型、包名、类名、方法名分类。按照原文中给出的系数，将 β_t 和 β_{AF} 均设为 0.5，系数 γ_{sf} 、 γ_c 和 γ_m 分别设置为 0.06、0.25 和 0.69。

$$ATA(a) = (\beta_t * t_{type}) + (\beta_{AF} * AF_{type}) \quad (5-3)$$

$$CL(a) = \beta_t * (\gamma_{sf} * t_{sf} + \gamma_c * t_c + \gamma_m * t_m) + \beta_{AF} * (\gamma_{sf} * AF_{sf} + \gamma_c * AF_c + \gamma_m * AF_m) \quad (5-4)$$

BaselineD

Haihao Shen 等人 [21] 从概率的角度出发计算缺陷可能性等统计数据，基于此提出有效的错误排序方案，以改进 FindBugs 中现有的排序方法。这里本文选择他们提出的缺陷模式可能性作为排序因子，详细计算步骤参见第四章警告特征提取中的警告分布特征一节。

BaselineE

Guangtai Liang 等人 [22] 使用了 K 最近邻分类器在训练集上训练，然后对收集的警告进行优先级排序，以精度评估时获得了不错的效果。他们使用的特征是本文所提取特征的一部分，因此本文也实现了一个 KNN 分类器，用于排序效果的对比。

5.1.2 数据集收集

为了能收集到质量较高的多版本的警告误报数据，本章首先确定从流行的开源项目中扫描获得警告。Apache 软件基金会是全球最大的开源社区，负责管理众多高质量的软件开源项目。这些项目大多数在 Github 上开源并会定期发布在 Maven 中央仓库之中。这里按照下列条件对项目进行初步的筛选：

- 项目的主要开发语言是 Java，并且能兼容 Java8；
- 项目未被归档，即项目仍在开发和使用中；
- 项目第一个版本的提交时间在 2019 年之前；
- 项目在近一年内仍有更新记录，即项目处于活跃状态。

经过筛选后，再综合其在 Github 获得 Star 的数量和近期版本更新的活跃程度，本文选择了表 5.1 列出的这些项目。

表 5.1: 项目详细信息

项目	总版本数	创建日期	更新日期	Github Star
hadoop-common	72	2014-08-28	2022-03-12	12396
zookeeper	38	2009-03-27	2022-03-12	10242
calcite-core	8	2014-06-25	2022-03-11	2966
pdfbox	9	2009-09-26	2022-03-10	1624
helix-core	10	2014-01-13	2022-03-11	361

确定了数据集的项目来源后，本章用 Python 实现了一个爬虫程序，访问 Maven 的中央仓库并从相关项目的 URL 中搜索各个版本。这里使用正则表达式“<a(.*)>(.*)(.*)\n”来匹配项目的版本和其对应的 URL。同时可以使用正则表达式“(.(+?)(2,)(.+?)”进一步匹配出该版本的提交时间和文件大小。在每个版本对应的 URL 中，可以匹配到一系列具有固定命名格式的 Jar 包，如表 5.2 所示，包含 sources 的是源代码打包，包含 javadoc 的是包含文档的代码包，直接以 jar 结尾的是可执行 Jar 包。最后下载 sources 包并解压获得源代码，下载对应的可执行 Jar 包作为编译打包后的产物。

表 5.2: Maven 仓库文件命名格式

文件名	含义
{name}-{version}.jar	编译后的字节码打包
{name}-{version}-sources.jar	Java 源文件打包
{name}-{version}-javadoc.jar	Javadoc 文档打包
{name}-{version}.pom	POM 描述文件

如果当前版本内 sources 包或 jar 包文件有缺失，则直接跳过这一版本。本文设定了一个时间范围，仅搜集从 2019 年 1 月 1 日到 2020 年 12 月 31 日之间发布的版本。对于被选中的版本，本文调用 wget 下载工具下载上文提到的两类

文件，sources 包在下载完成后调用 unzip 命令解压至指定的目录。对于每个版本的 jar 包，都使用 SpotBugs 工具扫描获得检测报告，并解析检测报告录入警告信息。接着横向比较各版本之间的警告，按照下列判断规则将警告分为正报和误报：

- 当前警告在之后的版本中不存在，标为正报；
- 当前警告因为文件删除不再存在，标为不确定；
- 剩余的警告标为误报。

最终本章搜集的项目信息如表 5.3 所示：

表 5.3: 项目警告详细信息

项目	可用版本数	警告总数	正报数目	误报数目
hadoop-common	8	7736	756	6785
zookeeper	8	3098	425	2330
calcite-core	8	7657	230	7222
pdfbox	9	6748	172	5802
helix-core	10	9459	865	8277

5.1.3 评估指标

对于一个共有 N 个样本的警告数据集，假设其中正报数量为 i ，误报数量为 j ，那么理想情况下的最优排序结果的标签序列 A_N 为：

$$A_1 = A_2 = \dots = A_i = 1, A_{i+1} = A_{i+2} = \dots = A_{i+j} = 0 \quad (5-5)$$

即所有的正报警告都被排在最前面，然后才是误报警告。本文选择使用以下三种指标：Fitness、Kendall 距离和 AUC，来衡量实验结果的标签序列 B_N 与最优序列 A_N 的差异值，以此评估实验模型的效果。

Fitness

假设实验结果序列中的一个样本标签 B_n ，其在最优序列中的位置为 A_m ，即本应排在 m 处的标签被排在了 n 处。对此用公式 5-6 计算 Fitness 指标的值，其取值范围在 0 到 1 之间，越接近 1 表明两个序列越接近，即效果更好。

$$Fitness_B = \frac{2}{N} \sum_{n=1}^N \frac{1}{|n-m|+1} \quad (5-6)$$

Kendall 距离

对于实验结果序列和最优序列里的任意两个样本标签 B_n 和 A_m ，定义 $k(m, n)$ 当且仅当 $A_m = 0$ 且 $B_n = 1$ 时取 1，其余情况下取 0。对此用公式 5-7 计算 Kendall 距离，其取值范围也在 0 到 1 之间，但是越接近 0 表明序列越接近。

$$Kendall_B = \frac{2}{N(N-1)} \sum_{m=1}^{N-1} \sum_{n>m}^N k(m, n) \quad (5-7)$$

AUC

当审核人员从前向后逐个审核警告时，已审核的正报数目不断增加，即已审核的正报占有所有正报警告的比例，随着已审核警告占比的增加而增加。以这两个变量为坐标轴可以绘制出它们之间的变化关系，曲线下与坐标轴所围的面积记为 AUC，可知其取值范围在 0 到 1 之间，但不会等于 0 或 1。AUC 的值越接近 1 说明实验效果越好，如果接近 0.5 则说明实验和随机排序的效果差不多。本文使用 sklearn 包下 metrics 中的 auc 方法来计算。

5.1.4 实验结果

RQ1 和其他警告排序方法相比，本文提出的基于主动学习的方法是否更有效？

RQ1 的实验结果如表 5.4 所示，可以看到在大部分项目中，使用工具自己生成的排序顺序的效果和随机排序的效果非常接近，这说明静态代码分析工具自身并不能很好地做出排序优化。在其他实验中，BaselineC 使用 ATA 和 CL 计算值排序能取得比较好的效果，而 BaselineE 使用 KNN 模型在不同项目上的效果并不稳定。本文所设计的基于主动学习的方法则均能取得不错的效果，AUC 值在 0.9 左右，部分可以接近 1 的水平。这说明本文提出的基于主动学习的排序优化方法比之前的方法效果更好。

RQ2 主动学习流程中，应用不同的机器学习对排序优化效果有何影响？

在 RQ2 中本文对比了不同机器学习模型的排序效果，包括集成学习模型中的 bagging 和 boosting 方法，以及传统机器学习模型中的支持向量机（svm）、随机森林（rt）、决策树（dt）、朴素贝叶斯（nb）和多层感知器（mlp）。

表 5.4: RQ1 的实验结果

项目	实验	Fitness	Kendell	AUC
calcite-core	BaselineA	0.2786	0.0050	0.4131
	BaselineB	0.3648	0.0031	0.6388
	BaselineC	0.4084	0.0017	0.8036
	BaselineD	0.3420	0.0039	0.5493
	BaselineE	0.2035	0.0116	0.4152
	AL	0.5000	0.0000	0.9970
hadoop-common	BaselineA	0.1184	0.0212	0.3652
	BaselineB	0.0999	0.0245	0.2696
	BaselineC	0.2662	0.0061	0.8128
	BaselineD	0.2171	0.0178	0.4663
	BaselineE	0.0686	0.0827	0.5816
	AL	0.1155	0.0575	0.5706
helix-core	BaselineA	0.0509	0.0755	0.5494
	BaselineB	0.0421	0.0871	0.4875
	BaselineC	0.1241	0.0430	0.7235
	BaselineD	0.1619	0.0702	0.5778
	BaselineE	0.1250	0.0847	0.5876
	AL	0.4224	0.0035	0.9194
pdfbox	BaselineA	0.0972	0.0339	0.4517
	BaselineB	0.1421	0.0224	0.6321
	BaselineC	0.2464	0.0165	0.7258
	BaselineD	0.1602	0.0313	0.4926
	BaselineE	0.3726	0.0065	0.8054
	AL	0.5000	0.0000	0.9760
zookeeper	BaselineA	0.1372	0.0648	0.5418
	BaselineB	0.1209	0.0720	0.4952
	BaselineC	0.3042	0.0324	0.7520
	BaselineD	0.1328	0.0841	0.4170
	BaselineE	0.0589	0.1149	0.2175
	AL	0.2849	0.0183	0.8635

表 5.5: RQ2 的实验结果

项目	实验	Fitness	Kendell	AUC
calcite-core	bagging	0.5000	0.0000	0.9970
	boosting	0.5000	0.0000	0.9971
	dt	0.5000	0.0000	0.9970
	mlp	0.5000	0.0000	0.9970
	nb	0.5000	0.0000	0.9971
	rt	0.5000	0.0000	0.9970
	svm	0.5000	0.0000	0.9970
hadoop-common	bagging	0.0776	0.0488	0.6173
	boosting	0.2546	0.0303	0.7589
	dt	0.0934	0.0589	0.5668
	mlp	0.2280	0.0256	0.7908
	nb	0.0624	0.0694	0.4936
	rt	0.1155	0.0575	0.5706
	svm	0.0818	0.0647	0.5212
helix-core	bagging	0.2275	0.0201	0.8584
	boosting	0.2352	0.0176	0.8673
	dt	0.1609	0.0260	0.8364
	mlp	0.2242	0.0236	0.8456
	nb	0.2013	0.0208	0.8556
	rt	0.4224	0.0035	0.9194
	svm	0.3895	0.0048	0.9143
pdfbox	bagging	0.5000	0.0000	0.9751
	boosting	0.5000	0.0000	0.9758
	dt	0.5000	0.0000	0.9757
	mlp	0.5000	0.0000	0.9752
	nb	0.5000	0.0000	0.9760
	rt	0.5000	0.0000	0.9760
	svm	0.5000	0.0000	0.9758
zookeeper	bagging	0.3672	0.0085	0.9069
	boosting	0.4065	0.0085	0.9059
	dt	0.3260	0.0124	0.8888
	mlp	0.3099	0.0161	0.8728
	nb	0.3437	0.0152	0.8768
	rt	0.2849	0.0183	0.8635
	svm	0.3611	0.0093	0.9020

RQ2 的实验结果如表 5.5 所示，综合来看集成学习的效果要优于传统的机器学习模型，不过传统的机器学习模型中仍有部分能取得不错的效果。

5.2 系统测试

系统测试是为了在用户开始使用软件前，尽可能地发现软件中潜在的错误和不合理之处，确保最后将高质量的软件交付给用户。为了验证系统的功能和性能是否与需求分析时的规格说明一致，本章设计并进行了功能测试、性能测试和用户界面测试。

5.2.1 测试环境

根据系统的物理部署视图，本章首先搭建了测试所用的系统环境。测试环境应当尽量模拟用户的真实使用环境和系统真实的运行环境。测试环境中用户机由自己的笔记本电脑担任，在 Windows 系统下运行 Chrome 浏览器，用于显示前端界面并与后端系统进行交互。为了保证测试环境的相对独立性，服务器由云主机担任，用于运行 Flask 后端服务器框架。由于测试环境的数据量并不大，因此同一服务器也兼任文件存储的功能，在真实生产环境中则需注意留下足够的磁盘可用空间。测试环境的具体物理配置信息如表 5.6 所示。

表 5.6: 测试环境物理配置

设备	配置项	详情
用户机	处理器	Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz
	内存	16G
	硬盘	1T
	系统	Windows 10 专业版 21H1
服务器	处理器	QEMU Virtual CPU version
	内存	4G
	硬盘	100G
	系统	Ubuntu 18.04 Server

为了顺利运行系统，服务器上需要配置系统运行的软件环境，详细信息如表 5.7 所示。除了 Python 包使用 pip 命令安装外，其余软件均可以通过 apt 命令从 Linux 软件仓库中安装。由于数据库使用了轻量级的 SQLite，因此不必额外

安装数据库的服务器程序。测试环境部署配置完成后应当能保证系统不会因为环境问题出错。

表 5.7: 测试环境服务器软件配置

配置项	详情
Java 运行环境	OpenJDK-8-jdk
Python 运行环境	Python 3.8
消息队列服务器	RabbitMQ Server
Linux 软件	wget unzip
Python 包	flask numpy pandas peewee pika requests sklearn wget

5.2.2 功能测试

功能测试是根据需求列表，验证产品实现的功能是否满足了需求规格说明的要求，是否有遗漏的功能点，以及能否正确地接收输入和输出结果。按照本文需求分析阶段给出的用例描述，以及概要设计阶段对系统模块的划分，本文设计并运行了以下的测试用例，覆盖了主要功能的正常流程和扩展流程。

表 5.8: 查看项目列表测试用例描述

ID	TC1
测试名称	查看项目列表测试
优先级	中
前置条件	管理员登录系统并通过认证
正常流程	1. 点击标题栏的“所有项目”按钮； 2. 查看系统中所有项目的信息； 3. 在下拉框中选择需要筛选的项目状态； 4. 点击“查看”按钮； 5. 查看处于特定状态的项目。
预期结果	1. 系统根据管理员选择的项目状态显示符合的项目列表，默认显示全部； 2. 系统展示各项目的名称、版本、描述、状态和已检测出的漏洞数量。
测试结果	通过

查看项目列表测试用例描述如表 5.8 所示，测试的是管理员查看项目列表的相关功能，主要测试点在于项目详情能以表格逐行展示，并且可用根据上方下

拉框中选择的项目状态筛选，方便用户的查看需求。

表 5.9: 项目上传测试用例描述

ID	TC2
测试名称	项目上传测试
优先级	高
前置条件	管理员登录系统并通过认证
正常流程	1. 点击标题栏的“项目上传”按钮； 2. 填写项目名称、项目版本和项目描述； 3. 选择需要上传的 zip 和 jar 文件； 4. 点击“上传”按钮。
预期结果	1. 系统根据执行结果显示“上传成功”或“上传失败”； 2. 系统将上传文件保存至指定目录； 3. 系统保存填写的项目信息； 4. 系统在后台启动了主动学习线程。
测试结果	通过

项目上传测试用例描述如表 5.9 所示，测试的是管理员项目上传的相关功能，主要测试点在于上传的文件能否正确保存到指定的路径，数据库正常新增记录，以及能返回提示性信息。注意在项目上传完毕之后，服务器会启动一个新的主动学习线程。

表 5.10: 查看最终报告测试用例描述

ID	TC3
测试名称	项目上传测试
优先级	高
前置条件	管理员登录系统并通过认证
正常流程	1. 点击标题栏的“查看报告”按钮； 2. 在下拉框中选择想要查看的项目和版本； 3. 点击“查看”按钮； 4. 查看此版本项目下各警告的详细信息。
预期结果	1. 系统在下拉框中显示所有审核结束的项目； 2. 系统展示各警告的类名、方法名、行号、路径、标记时间和标记结果。
测试结果	通过

查看最终报告测试用例描述如表 5.10 所示，测试的是管理员查看最终报告

的相关功能，主要测试点在于系统能否正确判断项目的当前状态并以此提供待选择的列表，以及项目下的警告信息是否查询正确并正常地在下面的表格中显示。

表 5.11: 查看审核任务测试用例描述

ID	TC4
测试名称	查看审核任务测试
优先级	中
前置条件	审核人员登录系统并通过认证
正常流程	1. 点击标题栏的“审核任务”按钮； 2. 查看目前被分配的所有审核任务的信息。
预期结果	1. 系统将展示结果按扫描时间倒序排列； 2. 系统展示各审核任务的项目名称、版本、警告类型、警告行号、扫描时间。
测试结果	通过

查看审核任务测试用例描述如表 5.11所示，测试的是审核人员查看自己被分配的审核任务功能，主要测试点在于系统能否正确展示被分配给当前登录的审核人员的审核任务列表。

表 5.12: 警告审核测试用例描述

ID	TC5
测试名称	警告审核测试
优先级	高
前置条件	审核人员登录系统并通过认证
正常流程	1. 点击审核任务后的“查看详情”按钮； 2. 查看审核任务的详细信息和相关的代码片段； 3. 选中“误报”或“正报”单选框； 4. 点击“确认提交”按钮。
预期结果	1. 系统展示审核任务的项目名称、项目版本、类名、方法名、行号、路径、源代码片段和模型对此的预测结果； 2. 系统保存审核人员的审核结果。
测试结果	通过

警告审核测试用例描述如表 5.12所示，测试的是审核人员保存对待审核警告的判断结果的功能，主要测试点在于系统能否正确显示指定审核任务的详细

信息，从源代码中切分读取相应的代码，以及保存审核人员的审核结果并反馈给主动学习模型。

表 5.13: 查看审核任务历史测试用例描述

ID	TC6
测试名称	查看审核任务历史测试
优先级	中
前置条件	审核人员登录系统并通过认证
正常流程	1. 点击标题栏的“审核历史”按钮； 2. 查看已经审核过的所有审核任务的信息。
预期结果	1. 系统将展示结果按审核时间正序排列； 2. 系统展示各审核任务的项目名称、版本、任务发布时间、实际审核时间和审核结果。
测试结果	通过

查看审核任务历史测试用例描述如表 5.13所示，测试的是审核人员查看自己已经审核过的审核任务功能，主要测试点在于系统能否正确展示当前登录的审核人员的历史审核任务列表。

以上测试用例的测试结果全部通过，达到了预期的结果，说明本系统实现了需求阶段所提出的功能性需求，能够满足用户的使用要求，为用户提供可用的服务，系统的开发达到了预期的效果和目的。

5.2.3 性能测试

性能测试是检查多用户同时在线时，服务器处理请求的效率是否会发生变化，以及是否存在其他时间敏感的问题。性能测试可以确认系统有无各种性能瓶颈，在多用户使用时能否正常运行而不发生故障。考虑到用户使用的频率和接口的重要程度，本文将需要测试的接口整理为表 5.14。

本文使用 Apache Benchmark 工具^①对接口进行测试，-c 选项设置并发数量为 50，-t 选项设置测试持续时间为 5s，-n 选项设置发送 100 个请求。对于 POST 请求测试时，需要将请求附带的数据存放在单独的文件内，测试时使用 -p 选项指定需要 POST 的数据。运行测试后的结果如表 5.15所示，说明系统运行时可以应对并发量较高的情况，在多用户同时使用时能保证正常可用的响应时间。

^①<https://httpd.apache.org/docs/2.4/programs/ab.html>

表 5.14: 性能测试接口列表

编号	接口路径	接口描述
API_1	/api/upload	项目上传
API_2	/api/project	查看项目列表
API_3	/api/report	查看最终报告
API_4	/api/alarm	查看审核任务列表
API_5	/api/alarm/{alarm_id}	查看审核任务详情
API_6	/api/check/{alarm_id}	提交审核结果
API_7	/api/history	查看历史审核记录列表

表 5.15: 性能测试接口列表

编号	错误请求数量	平均响应时间	99% 响应时间
API_1	0	45ms	97ms
API_2	0	38ms	102ms
API_3	0	17ms	56ms
API_4	0	21ms	78ms
API_5	0	19ms	74ms
API_6	0	28ms	81ms
API_7	0	23ms	77ms

5.2.4 用户界面测试

用户界面测试是检查系统的各个页面是否符合人机交互的基本设计规范，是否达到了可用性的要求，比如各界面风格是否统一，提示信息是否全面，是否符合用户的操作习惯。本文所实现的系统界面数量有限，跳转逻辑也并不复杂，因此本文选择采用手动测试的方式。用户界面测试可以与功能测试同步进行，本节给出了测试时部分系统界面的截图。

图 5.1为系统管理员查看项目列表功能的截图，对应需求用例 UC1 和测试用例 TC1。本页面作为管理员登录后展示的首页，展示已上传各项目的详细信息，包括名称、版本、描述、状态和已检测出的漏洞数量。同时，通过下拉框可以根据项目当前状态筛选，方便管理员查看。

图 5.2为系统管理员上传项目功能的截图，对应需求用例 UC2 和测试用例 TC2。本页面为管理员上传项目提供了信息填写和文件选择的功能，包括项目的名称、版本号和描述，需要选择 zip 文件和 jar 文件，点击“上传”按钮后可

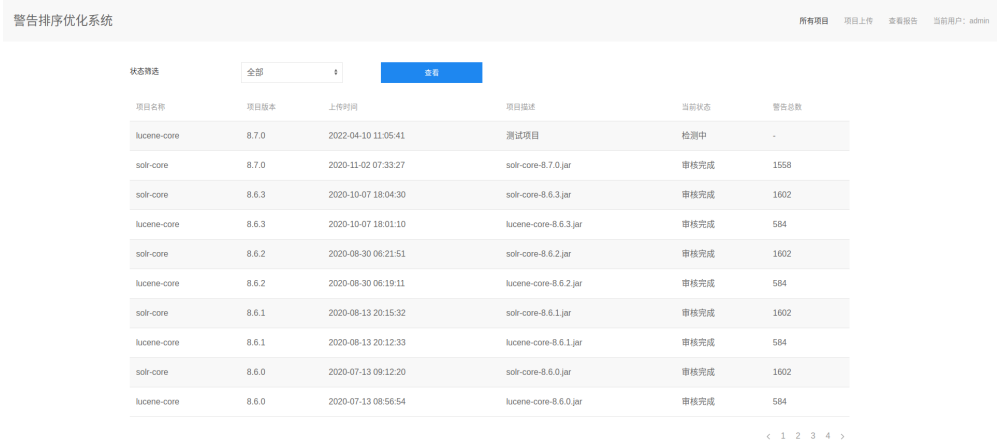


图 5.1: 查看项目列表页面



图 5.2: 项目上传页面

以显示上传操作的执行结果。

图 5.3为系统管理员查看最终报告功能的截图，对应需求用例 UC3 和测试用例 TC3。本页面提供了下拉列表框供管理员选择需要查看最终审核结果报告的项目，并展示项目内各警告的详细信息，包括位置、类、方法、审核结果等，在表格中优先显示正报结果，并提供筛选按钮。

图 5.4为审核人员查看审核任务功能的截图，对应需求用例 UC4 和测试用例 TC4。本页面用于向登录后的审核人员展示当前被分配的审核任务，以及审

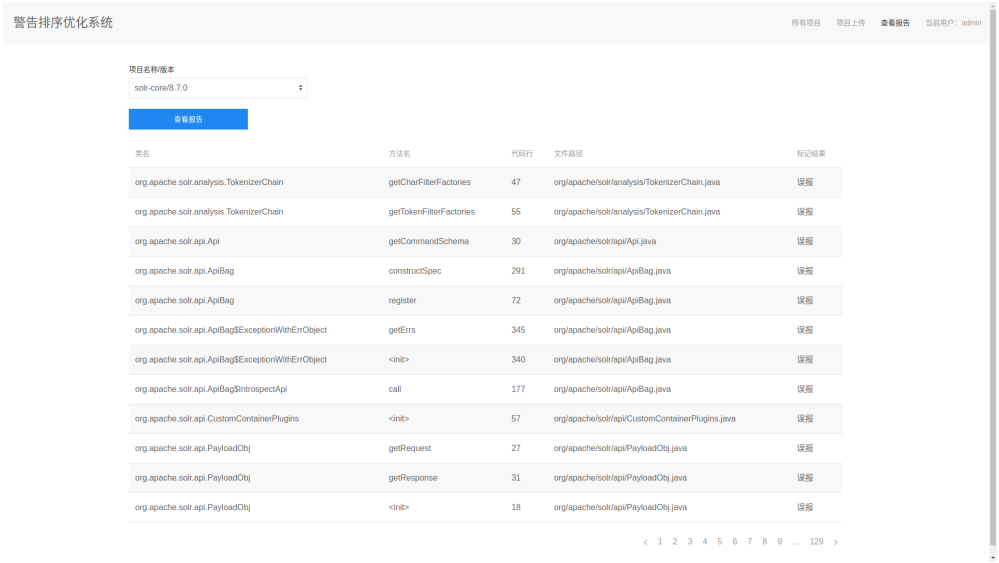


图 5.3: 查看最终报告页面

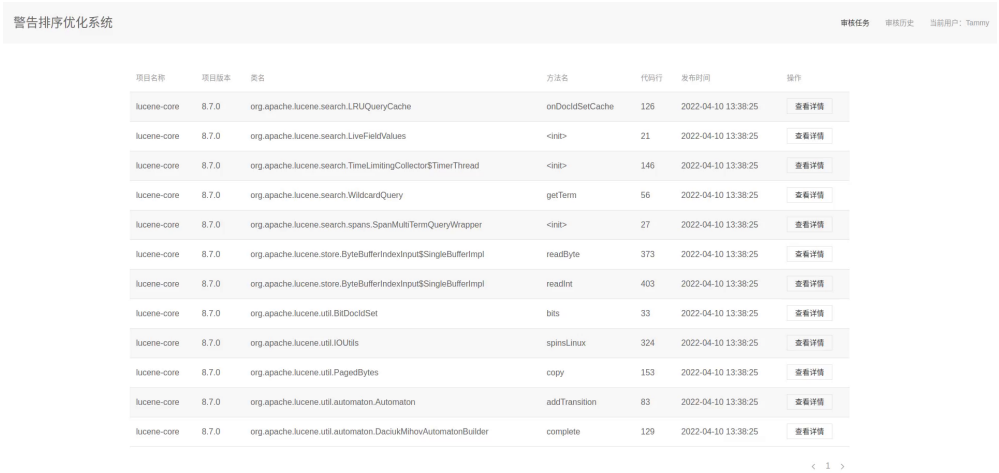


图 5.4: 查看审核任务列表页面

核任务的概要信息，包括其对应项目的名称、版本号等。本页面将优先展示最先被发布的审核任务，以提示审核人员及时处理。

图 5.5 为审核人员填写审核结果功能的截图，对应需求用例 UC5 和测试用例 TC5。本页面用于展示一个审核任务的详细信息，包括项目名称、项目版本、代码类名、方法名、警告分类、警告类型、方法签名和代码片段，其中代码片段中用高亮标出了报告出警告的代码行。审核人员在最下方选择审核的结果，并点击“确认提交”按钮提交审核结果。

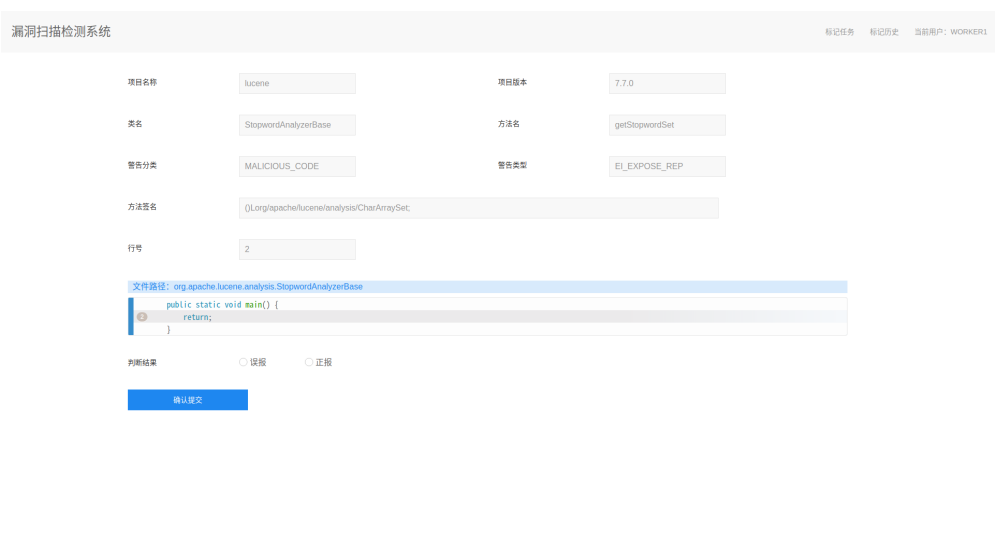


图 5.5: 查看审核任务详情页面

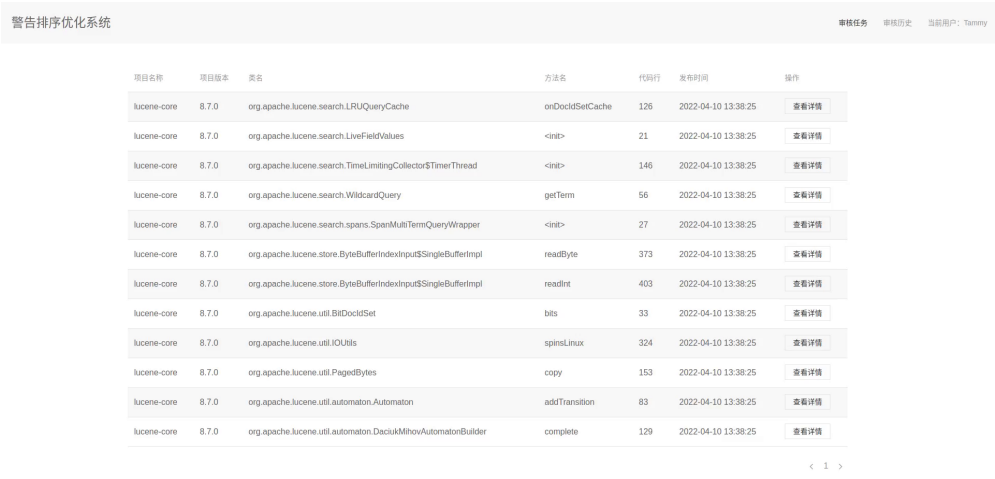


图 5.6: 查看审核任务历史页面

图 5.6为系统管理员上传项目功能的截图，对应需求用例 UC6 和测试用例 TC6。本页面用于展示历史审核任务的标记情况，包括项目名称、项目版本、任务发布时间、标记时间和标记结果。

5.3 本章小结

本章通过收集构建数据集并设计了多个实验，分别证明了引入主动学习技术的排序方法效果优于当前的排序方法，同时主动学习内部集成学习模型的使用也提升了排序的效果。同时本章搭建测试环境，在测试服务器上对本文实现的基于主动学习的警告排序系统进行了测试。测试结果表明本文的系统实现了需求规格说明中确定的各项业务需求，在多用户高并发的情况下也可正常使用，保证了系统一定程度上的可靠性和可用性。作为系统测试的一部分，本章也给出了用户界面的实际运行截图。

第六章 总结与展望

6.1 总结

静态代码分析工具能在软件开发过程中及早地发现缺陷，降低软件开发和测试的成本，因此被越来越多的企业和开发者应用于实际开发中。但是目前的静态代码分析工具在应用中存在着不少的问题，其中最突出的一点就是，检测报告中的警告误报率过高，即有很多都不是真正的缺陷，使得测试人员必须再次花费时间和精力对警告进行二次审核。这是因为静态分析技术并不会真的执行程序，并且设计的初衷是为了尽可能地找出可疑的代码而不过多地考虑准确性。在当前的改进方案中，有研究使用机器学习技术识别警告中的误报，但是机器学习技术需要大量的训练数据，并且难以训练出相对通用的模型用在每一个项目上。也有研究采用排序的方法，但是这些排序方法大多数是一次性的，以版本为单位才能改进排序的效果，难以及时地利用上审核人员的反馈结果。

本文针对以上问题，提出了基于主动学习的警告排序技术，开发了带有排序功能的警告检测审核系统。在一个 Java 项目的实际开发过程中，每当完成一个版本的开发时，项目的负责人可以编译打包项目，将 Jar 包和源代码上传到系统。之后系统将自动运行静态代码分析工具对其检测，并启动设定好的主动学习流程，将警告分配给不同的审核人员进行二次审核。这其中的核心是主动学习技术，不断从未审核的警告中选取警告分发给审核人员审核，并更新审核结果重新训练模型，以此达到持续改进模型训练效果的作用。如果模型的预测效果达到了一定的水平，将不再分配审核任务，以此减轻审核人员的工作负担。此外，之前版本的警告训练的模型将会继续被用于下一版本的训练中，一定程度上考虑到了项目各版本之间的连续性。审核人员查看被分配的审核任务，阅读警告的相关信息和源代码，并参考主动学习模型给出的预测结果，做出自己的判断并反馈给系统。在主动学习流程结束后，项目负责人可以查看检测以及审核的结果，据此做出下一步的处理和工作安排。

本文首先介绍了本开发项目的背景与意义，分析了当前警告误报检测技术的研究现状，说明了在现有工作的启发下本文的主要工作内容，接着介绍了系

统实现中涉及的相关技术和代码开发框架。在第三章，本文描述了系统需求分析和概要设计的过程和内容，使用用例图和用例规格描述构建需求模型，使用4+1视图描述系统的整体结构，完成了模块划分并借助概念类图、流程图和时序图描述各个模块的详细设计。然后在第四章给出了各模块实现过程中的核心代码，尤其是对警告和相关联源代码的特征提取以及主动学习流程，做出了较为详细的介绍。最后本文基于 Apache 开源项目构建警告误报数据集，对本文提出的基于主动学习的警告排序算法进行了实验验证。另一方面针对需求中提出的功能性需求和非功能性需求设计了测试用例，测试了本文所开发的系统的实际运行效果，并附上了系统关键功能的运行截图。

6.2 展望

本文所提出了基于主动学习技术的警告优化系统实现了对可操作警告的及时识别，减少了审核人员的工作量，使得项目开发迭代过程中能够更加关注于可操作漏洞本身的修复处理，满足了项目开发的基本需求。不过系统的实现仍然有许多需要进一步改进提升的地方，主要有以下几点：

1. 警告的特征不够丰富。虽然现在使用的特征兼顾了警告和其相关联的源代码片段两方面，但是项目开发过程中在版本控制系统上的提交情况、代码文件的变化情况等其他角度也可以获得丰富的特征。未来特征提取模块可以得到进一步扩展和更新。
2. 未考虑审核人员的审核效果。本文的系统默认审核人员均能较为准确地判断警告是否为待修复的真实漏洞，但是实际情况可能并非如此。因此可能需要将同一个警告分配给多个审核人员审核，综合判断审核的结果，同时还要考虑不同审核人员的审核准确率。
3. 对于跨项目的支持不好。本文所设计的流程只能在同一项目的不同版本中达到预期的功能。但是实际应用中，团队开发的不止一个项目，如何总结出跨项目可使用的先验知识就是系统后续需要解决的问题。理想情况下系统中的模型应当能从不同的软件项目中学习得出一般的判断规则。

综上所述，本文所设计并实现的基于主动学习的警告排序系统仍然存在着提升空间，对此后续的改进中可以考虑引入新的特征，或是引入对审核人员的激励和投票策略，或是进一步设计并完善机器学习模型，以达到进一步提升系统功能可用性的目的。

致 谢

大江之东，金城之中，天开教泽，道启南雍。余幸得于此求学六载，怎料韶光易逝，岁月匆匆，寒来暑往，恍然一梦。此间春花烂漫，秋桂飘香，夏日荫浓，冬雪银装，四时景致，各有所长。更有廊前屋下，狸奴相戏，梁上檐侧，倦鸟长依，光景甚好，无问今昔。不想花开花谢，终至别时，顿足回首，流年成诗。恰逢母校百廿生辰，愿其于市井之中守宁静，于浮华之下留清辉，不负百年声名。

导师房春荣先生，自选题始，至定稿终，耳提面命，指点良多。实验室组内葛修婷学姐，悉心指导，勤加勉励，文章不足之处均一一指出。二位不辞辛劳，余不胜感激，在此拜谢。

余寝室四人有缘相聚于此，虽共度不过两轮寒暑，然融洽相处，欢声笑语常驻。惜至分别之时，各赴前路，在此遥祝，诸位往后，前程似锦，一帆风顺。

求学之路有终，为人之道无穷。毕业此去，辗转于高楼之中，栖身于他人之所，纵无近虑，却有远忧。正可谓，鸿鹄纵有凌云志，青天几重愴平生。然舟行至此，何须退路，激流勇进，争赶潮头，以毕生所学，斩荆棘，暴霜露，不负年华，不辞辛苦。纵拼搏正当时，亦勿忘注意身体康健，以便他岁会旧友，把酒再言欢。

别了，吾校，别了，吾友。

是岁壬寅年三月

作于鼓楼南园

参考文献

- [1] ARTHO C, SCHUPPAN V, BIERE A, et al. JNuke: Efficient dynamic analysis for Java[C] // International Conference on Computer Aided Verification. 2004 : 462 – 465.
- [2] VAN HOORN A, WALLER J, HASSELBRING W. Kieker: A framework for application performance monitoring and dynamic software analysis[C] // Proceedings of the 3rd ACM/SPEC international conference on performance engineering. 2012 : 247 – 248.
- [3] AYEWAH N, PUGH W, HOVEMEYER D, et al. Using static analysis to find bugs[J]. IEEE software, 2008, 25(5): 22 – 29.
- [4] JOHNSON B, SONG Y, MURPHY-HILL E, et al. Why don't software developers use static analysis tools to find bugs?[C] // 2013 35th International Conference on Software Engineering (ICSE). 2013 : 672 – 681.
- [5] KIM S, ERNST M D. Which warnings should I fix first?[C] // Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2007 : 45 – 54.
- [6] RUTHRUFF J, PENIX J, MORGENTHALER J, et al. Predicting accurate and actionable static analysis warnings[C] // 2008 ACM/IEEE 30th International Conference on Software Engineering. 2008 : 341 – 350.
- [7] YANG X, CHEN J, YEDIDA R, et al. Learning to recognize actionable static code warnings (is intrinsically easy)[J]. Empirical Software Engineering, 2021, 26(3): 1 – 24.
- [8] KANG H J, AW K L, LO D. Detecting False Alarms from Automatic Static Analysis Tools: How Far are We?[J]. arXiv preprint arXiv:2202.05982, 2022.

- [9] KREMENEK T, ENGLER D. Z-ranking: Using statistical analysis to counter the impact of static analysis approximations[C] // International Static Analysis Symposium. 2003 : 295 – 315.
- [10] KREMENEK T, ASHCRAFT K, YANG J, et al. Correlation exploitation in error ranking[J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(6) : 83 – 93.
- [11] LI M, ZHANG H, WU R, et al. Sample-based software defect prediction with active and semi-supervised learning[J]. Automated Software Engineering, 2012, 19(2) : 201 – 230.
- [12] MURTAZA S S, KHREICH W, HAMOU-LHADJ A, et al. Mining trends and patterns of software vulnerabilities[J]. Journal of Systems and Software, 2016, 117 : 218 – 228.
- [13] HOVEMEYER D, PUGH W. Finding bugs is easy[J]. Acm sigplan notices, 2004, 39(12) : 92 – 106.
- [14] THUNG F, LO D, JIANG L, et al. To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools[C] // 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012 : 50 – 59.
- [15] MARCILIO D, BONIFÁCIO R, MONTEIRO E, et al. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube[C] // 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC). 2019 : 209 – 219.
- [16] VASSALLO C, PANICHELLA S, PALOMBA F, et al. How developers engage with static analysis tools in different contexts[J]. Empirical Software Engineering, 2020, 25(2) : 1419 – 1457.
- [17] NAM J, WANG S, XI Y, et al. Designing bug detection rules for fewer false alarms[C] // Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings. 2018 : 315 – 316.

- [18] HANAM Q, TAN L, HOLMES R, et al. Finding patterns in static analysis alerts: improving actionable alert ranking[C] // Proceedings of the 11th working conference on mining software repositories. 2014 : 152 – 161.
- [19] HEO K, OH H, YI K. Machine-learning-guided selectively unsound static analysis[C] // 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). 2017 : 519 – 529.
- [20] SHIVAJI S, WHITEHEAD E J, AKELLA R, et al. Reducing features to improve bug prediction[C] // 2009 IEEE/ACM International Conference on Automated Software Engineering. 2009 : 600 – 604.
- [21] SHEN H, FANG J, ZHAO J. Efindbugs: Effective error ranking for findbugs[C] // 2011 Fourth IEEE International conference on software testing, verification and validation. 2011 : 299 – 308.
- [22] LIANG G, WU L, WU Q, et al. Automatic construction of an effective training set for prioritizing static analysis warnings[C] // Proceedings of the IEEE/ACM international conference on Automated software engineering. 2010 : 93 – 102.
- [23] ALLIER S, ANQUETIL N, HORA A, et al. A framework to compare alert ranking algorithms[C] // 2012 19th Working conference on reverse engineering. 2012 : 277 – 285.
- [24] 毛蔚轩, 蔡忠闽, 童力. 一种基于主动学习的恶意代码检测方法 [J]. 软件学报, 2017, 28(2): 14.
- [25] 李延超、肖甫、陈志、李博. 自适应主动半监督学习方法 [J]. 软件学报, 2020, 31(12): 15.
- [26] YU Z, MENZIES T. Total recall, language processing, and software engineering[C] // Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering. 2018 : 10 – 13.
- [27] YU Z, THEISEN C, WILLIAMS L, et al. Improving vulnerability inspection efficiency using active learning[J]. IEEE Transactions on Software Engineering, 2019.

- [28] WANG J, WANG S, CUI Q, et al. Local-based active classification of test report to assist crowdsourced testing[C] // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016 : 190–201.
- [29] YANG X, MENZIES T. Documenting evidence of a reproduction of ‘is there a “golden” feature set for static warning identification?—an experimental evaluation’ [C] // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021 : 1603–1603.
- [30] SETTLES B. Active learning literature survey[J], 2009.
- [31] ANGLUIN D. Queries and concept learning[J]. Machine learning, 1988, 2(4): 319–342.
- [32] COHN D, ATLAS L, LADNER R. Improving generalization with active learning[J]. Machine learning, 1994, 15(2): 201–221.
- [33] LEWIS D D, GALE W A. A sequential algorithm for training text classifiers[C] // SIGIR ' 94. 1994 : 3–12.
- [34] SHANNON C E. A mathematical theory of communication[J]. The Bell system technical journal, 1948, 27(3): 379–423.
- [35] CULOTTA A, MCCALLUM A. Reducing labeling effort for structured prediction tasks[C] // AAAI : Vol 5. 2005 : 746–751.
- [36] SEUNG H S, OPPER M, SOMPOLINSKY H. Query by committee[C] // Proceedings of the fifth annual workshop on Computational learning theory. 1992 : 287–294.
- [37] SETTLES B, CRAVEN M, RAY S. Multiple-instance active learning[J]. Advances in neural information processing systems, 2007, 20.
- [38] COHN D A, GHAHRAMANI Z, JORDAN M I. Active learning with statistical models[J]. Journal of artificial intelligence research, 1996, 4 : 129–145.
- [39] ZHANG T, OLES F. A probability analysis on the value of unlabeled data for classification problems. 17th ICML (pp. 1191–1198)[J], 2000.

-
- [40] BREIMAN L. Bagging predictors[J]. Machine learning, 1996, 24(2): 123 – 140.
 - [41] FREUND Y, SCHAPIRE R E. A decision-theoretic generalization of on-line learning and an application to boosting[J]. Journal of computer and system sciences, 1997, 55(1): 119 – 139.
 - [42] WOLPERT D H. Stacked generalization[J]. Neural networks, 1992, 5(2): 241 – 259.
 - [43] WANG J, WANG S, WANG Q. Is there a” golden” feature set for static warning identification? an experimental evaluation[C] //Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement. 2018: 1 – 10.
 - [44] HECKMAN S S, WILLIAMS L A. A measurement framework of alert characteristics for false positive mitigation models[R]. [S.l.]: North Carolina State University. Dept. of Computer Science, 2008.
 - [45] HECKMAN S, WILLIAMS L. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques[C] //Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. 2008: 41 – 50.
 - [46] HECKMAN S, WILLIAMS L. A model building process for identifying actionable static analysis alerts[C] // 2009 International conference on software testing verification and validation. 2009: 161 – 170.
 - [47] GRINBERG M. Flask web development: developing web applications with python[M]. [S.l.]: ” O’Reilly Media, Inc.”, 2018.
 - [48] FERNANDES J L, LOPES I C, RODRIGUES J J, et al. Performance evaluation of RESTful web services and AMQP protocol[C] // 2013 Fifth international conference on ubiquitous and future networks (ICUFN). 2013: 810 – 815.
 - [49] ROMAN G-C. A taxonomy of current issues in requirements engineering[J]. IEEE computer, 1985, 18(4): 14 – 23.

-
- [50] 骆斌. 需求工程: 软件建模与分析 (第二版)[M]. [S.l.]: 高等教育出版社, 2015: 40–41.

简历与科研成果

基本信息

钱美缘，男，汉族，1999 年 4 月出生，江苏省淮安人。

教育背景

2016 年 9 月 — 2020 年 6 月 南京大学软件学院软件工程专业 本科

攻读工程硕士学位期间完成的学术成果

1. Xiuting Ge, Chunrong Fang, Meiyuan Qian, Yu Ge, Mingshuang Qing, “Locality-based security bug report identification via active learning,” in *Information and Software Technology (2022)*, 2022.
2. Meiyuan Qian, Jun Luo, Yu Ge, Chen Sun, Xiuting Ge, Wanmin Huang, “Semantic-based false alarm detection approach via machine learning,” in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2021